Amazon GameLift

FlexMatch 开发人员指南

版本

Amazon GameLift: FlexMatch 开发人员指南

Table of Contents

什么是 GameLift FlexMatch?

GameLift FlexMatch 是一项针对多人游戏的可自定义对战服务。借助 FlexMatch，您可以构建一组自定义的规则，定义多人游戏的匹配情况，并确定如何为每个对战游戏评估和选择兼容的玩家。您还可以自定义对战过程的主要方面，包括微调匹配算法，以适应您的游戏。

FlexMatch 可与 GameLift 游戏托管解决方案（包括 实时服务器）一起使用，也可以用作独立的对战服务。您可以通过使用对等架构的游戏、在本地或其他云计算解决方案（包括 FlexMatch）上托管游戏服务器，将 GameLift FleetIQ 作为独立功能实施。本指南提供了有关如何为任何这些场景构建对战系统的详细信息。

FlexMatch 让您能够根据游戏要求灵活地设置对战优先级。例如，您可以执行以下操作：

- 在匹配速度和质量之间找到平衡。设置对战游戏规则以快速查找足够良好的对战游戏，或允许玩家等待一段时间以找到最佳玩家体验的最佳对战游戏。
- 根据对战的玩家或对战的团队进行对战。创建一个对所有玩家都具有非常相似特征（如技能或经验）的对战游戏。或者，如果每个团队的组合特征相似，表单化匹配，即使单个玩家的特征变化更大。
- 确定玩家延迟因素的优先级。为对战游戏中的所有玩家设置硬性延迟限制，或确保对战游戏的每个人的延迟非常相似，或者同时设置两者。

准备好开始使用 FlexMatch?

按照这些指南进行操作，以获得有关使用 FlexMatch 启动和运行游戏的分步指南：指南。

- FlexMatch 与 GameLift 托管的集成 (p. 8)
- GameLift 用于单独对战的 FlexMatch 集成 (p. 7)

主要 FlexMatch 功能

以下功能可用于所有 FlexMatch 场景，无论其是用作独立服务还是与 GameLift 游戏托管一起使用。

- 可自定义的玩家匹配。 设计和构建对战构建器以适合您为玩家提供的所有游戏模式。构建一组自定义规则来评估关键玩家属性（如技能级别或角色）和地理延迟数据，从而为游戏构成非常出色的玩家对战。
- 基于延迟的匹配。 提供玩家延迟数据并创建要求对战中的玩家具有类似响应时间的对战游戏规则。当您的玩家对战池跨多个地理区域时，此功能非常有用。
- 支持最多 200 位玩家的对战游戏。 使用为您的游戏自定义的对战规则创建最多 40 位玩家的对战游戏。使用匹配流程创建最多 200 位玩家的对战，该流程使用简化的自定义匹配流程保持玩家等待时间可管理。
- 玩家接受。 要求玩家在最终确定对战游戏并启动游戏会话前选择加入建议的对战游戏。使用此功能启动您的自定义接受工作流程，并在为对战游戏放置新游戏会话之前报告玩家对 FlexMatch 的响应。如果并非所有玩家接受对战游戏，则建议的对战游戏失败，接受的玩家将自动返回到对战池中。

  玩家组队支持。 为要组队一起玩游戏的玩家组生成对战游戏。使用 FlexMatch 查找更多玩家来根据需要填充对战。
- 可扩展的匹配规则。 在经过了一定的时间之后，逐步放宽对战游戏要求，而不会找到成功的对战游戏。规则扩展可让您确定何时何处放松初始对战规则，以便玩家可在合理时间段内进入可玩的游戏。
- 对战回填。 使用匹配良好的新玩家填充现有游戏会话中的空玩家位置。自定义何时以及如何请求新玩家，并使用相同的自定义对战规则来查找其他玩家。

使用 FlexMatch 托管的GameLift

对于通过 GameLift 托管的游戏，FlexMatch 提供以下附加功能。在使用 GameLift 服务托管自定义游戏服务器或使用 实时服务器 时，可以使用这些功能。使用 Amazon EC2 托管在 GameLift FleetIQ 资源上的游戏必须实施 FlexMatch 作为独立功能。

- 游戏会话放置。 成功进行对战游戏后，FlexMatch 会自动从 GameLift 请求新的游戏会话放置。对战游戏期间生成的数据（包括玩家 IDs 和团队分配）会提供给游戏服务器，因此，游戏服务器可以使用该信息为对战游戏启动游戏会话。然后，FlexMatch 传递回游戏会话连接信息，以便游戏客户端可以加入游戏。使用 GameLift 的游戏会话放置还可以使用区域玩家延迟数据（如果提供），以最大程度地减少对战游戏中的所有玩家遇到的延迟。
- 自动对战回填。 启用此功能后，当新游戏会话开始时，FlexMatch 会自动发送对战回填请求并且有未填充的玩家位置。使用此功能，您的对战系统将使用最少数量的玩家启动游戏会话放置过程，然后快速填充剩余位置。自动回填不能用于替换退出已对战游戏会话的玩家。

Amazon GameLift FlexMatch 的工作原理

本主题概述了 GameLift FlexMatch 服务，包括 FlexMatch 系统的核心组件以及它们的交互方式。

您可以将 FlexMatch 与使用 GameLift 托管托管托管托管托管的游戏或使用其他托管解决方案的游戏结合使用。在 GameLift 服务器（包括 实时服务器）上托管的游戏使用集成的 GameLift 服务自动找到可用的游戏服务器并为对战游戏启动游戏会话。将 FlexMatch 作为独立服务的游戏（包括 GameLift FleetIQ）必须与现有托管系统协调，才能分配托管资源并为对战游戏启动游戏会话。

有关为游戏设置 FlexMatch 的详细指导，请参阅

使用本节中的资源可帮助您开始使用 FlexMatch 构建对战系统。

主题

GameLift 用于单独对战的 FlexMatch 集成

本主题概述了将 FlexMatch 作为独立对战服务实施的完整集成过程。如果您的多人游戏是使用对等连接、自定义本地配置的硬件或其他云计算基元托管的，则使用此过程。此过程还可与 GameLift FleetIQ 一起使用，后者是适用于托管在 Amazon EC2 上的游戏的托管优化解决方案。如果您使用 GameLift 托管托管托管托管（包括 实时服务器）来托管游戏，请参阅FlexMatch 与 GameLift 托管的集成 (p. 8)。

在开始集成之前，您必须拥有 AWS 账户并设置 GameLift 服务的访问权限。有关详细信息，请参阅 设置 FlexMatch (p. 6)。与创建和管理 GameLift FlexMatch 对战构建器和规则集相关的所有关键任务都可以使用 Amazon GameLift 控制台完成，但您可能还需要 获取并安装 AWS Command Line Interface 工具。。

1. 创建 FlexMatch 对战规则集。 您的自定义规则集提供有关如何构建对战的完整说明。在其中，您可以定义每个团队的结构和规模。您还需要提供对战必须满足才有效的一组要求，FlexMatch 使用该要求在对战中包括或排除玩家。这些要求可能适用于单个玩家。您还可以自定义规则集中的 FlexMatch 算法，例如，与最多 200 位玩家构建大型对战游戏。请参阅这些主题：
   - 构建 FlexMatch 规则集 (p. 13)

    - FlexMatch 规则集示例 (p. 22)
2. 设置对战事件的通知。 使用通知跟踪 FlexMatch 对战活动，包括待处理对战请求的状态。这是用于提供建议的匹配项结果的机制。由于对战请求是异步的，您需要通过某种方式跟踪请求状态。使用通知是此项的首选选项。请参阅这些主题：
    - 设置 FlexMatch 事件通知 (p. 34)
    - FlexMatch 对战事件 (p. 58)
3. 设置 FlexMatch 对战配置。 此组件也称为对战构建器，它接收对战请求并对其进行处理。您可以通过指定规则集、通知目标和最长等待时间来配置对战构建器。您还可以启用可选功能。请参阅这些主题：
    - 设计 FlexMatch 对战构建器 (p. 9)
    - 创建对战配置 (p. 11)
4. 构建客户端对战服务。 创建或扩展具有生成对战请求并将其发送到 FlexMatch 的功能的游戏客户端服务。要构建对战请求，此组件必须具有相应的机制来获取对战规则集所需的玩家数据，以及（可选）区域延迟信息。它还必须具有为每个请求创建和分配唯一票证 IDs 的方法。您还可以选择构建要求玩家选择加入建议的对战游戏的玩家接受工作流程。请参阅以下主题：
    - 将 FlexMatch 添加到游戏客户端 (p. 36)
5. 构建对战放置服务。创建与您现有游戏托管系统配合使用的机制，以查找可用的托管资源并为对战启动新游戏会话。此组件必须能够检索 FlexMatch 对战游戏结果，并使用该信息将对战游戏与可用游戏服务器匹配，并指导它为对战设置新的游戏会话。您可能还希望实施一个工作流程以发出对战回填请求，该请求使用对战来填充已在运行的已对战游戏会话中的开放位置。

FlexMatch 与 GameLift 托管的集成

FlexMatch 适用于托管 GameLift，用于托管自定义游戏服务器和 实时服务器。要为您的游戏添加 FlexMatch 对战，请完成以下任务。

- 设置对战构建器。 对战构建器会接收玩家的对战请求并进行处理。它根据一组定义的规则将玩家分组，并为每个成功的对战游戏创建新的游戏会话和玩家会话。请按照以下步骤设置对战构建器：
    - 创建规则集。 规则集可以让对战构建器了解如何构建有效的对战游戏。它指定团队结构，并指定如何评估玩家是否参加对战游戏。请参阅这些主题：
        - 构建 FlexMatch 规则集 (p. 13)
        - FlexMatch 规则集示例 (p. 22)
    - 创建游戏会话队列。 队列查找每个对战游戏的最佳区域，并在该区域中创建新的游戏会话。使用现有队列或者为对战创建一个新队列。请参阅以下主题：
        - 创建队列
    - 设置通知 (可选)。 由于对战请求是异步的，您需要通过某种方式跟踪请求状态。通知是首选选项。请参阅以下主题：
        - 设置 FlexMatch 事件通知 (p. 34)
    - 配置一个对战构建器。 一旦您拥有了规则集、队列和通知目标，即可为您的对战构建器创建配置。请参阅这些主题：
        - 设计 FlexMatch 对战构建器 (p. 9)
        - 创建对战配置 (p. 11)
- 将 FlexMatch 集成到您的游戏客户端服务中。 向游戏客户端服务添加功能来启动包含对战的新游戏会话。对战请求指定要使用的对战构建器并为对战提供必要的玩家数据。请参阅以下主题：

- 将 FlexMatch 添加到游戏客户端 (p. 36)

- 将 FlexMatch 集成到您的游戏服务器。 向游戏服务器添加功能来启动通过对战创建的游戏会话。此类游戏会话的请求包括对战特定信息，其中包括玩家和团队分配。在为对战构建游戏会话时，游戏服务器需要访问和使用这些信息。请参阅以下主题：
  - 将 FlexMatch 添加到 GameLift 托管的游戏服务器 (p. 40)
- 设置 FlexMatch 回填（可选）。 请求更多玩家匹配来填充现有游戏中空闲的玩家位置。您可以打开自动回填来让 GameLift 管理回填请求。或者，您可以通过向游戏客户端服务或游戏服务器添加功能以发起对战回填请求来手动管理回填。请参阅以下主题：
  - 使用 FlexMatch 回填现有游戏 (p. 41)

    Note

    FlexMatch 回填当前对使用 实时服务器 的游戏不可用。

(p. 7)。

对战组件

对战系统包括以下部分或全部组件。FlexMatch

GameLift 组件

这些是控制 GameLift 服务如何为您的游戏执行对战的 FlexMatch 资源。它们是使用 GameLift 工具（包括控制台和 AWS CLI）创建和维护的，或者以编程方式使用适用于 GameLift 的 AWS 开发工具包创建和维护的。

- FlexMatch 对战配置（也称为对战构建器）对战构建器是为游戏自定义对战过程的一组配置值。–一个游戏可以有多个对战构建器，每个对战构建器针对所需的不同游戏模式或体验进行配置。当您的游戏向 FlexMatch 发送对战请求时，它会指定要使用的对战构建器。
- FlexMatch 对战规则集 – 规则集包含为潜在对战游戏评估玩家并批准或拒绝所需的所有信息。规则集定义对战游戏的团队结构，声明用于评估的玩家属性，并提供描述可接受对战游戏标准的规则。规则可应用于单个玩家、团队或整个对战游戏。例如，规则可能要求对战游戏中的每个玩家选择相同的游戏地图，也可能要求所有团队具有相似的玩家技能平均值。
- GameLift 游戏会话队列（对于仅具有 FlexMatch 托管托管托管的 GameLift）游戏会话队列查找可用的托管资源并为对战游戏启动新的游戏会话。–队列的配置确定 GameLift 查找可用托管资源的位置以及如何为对战选择最佳可用主机。

自定义组件

以下组件包含完整 FlexMatch 系统所需的功能，您必须根据游戏架构实施该功能。

- 用于对战的玩家界面 此界面使玩家能够加入对战游戏。–至少，它通过客户端对战服务组件发起对战请求，并根据对战过程提供特定于玩家的数据，例如技能级别和延迟数据。

  Note

  作为最佳实践，与 FlexMatch 服务的通信应由后端服务完成，而不是来自游戏客户端。
- 客户端对战服务 此服务用于填写玩家从玩家界面加入请求，生成对战请求，并将其发送到 – 服务的字段。FlexMatch对于正在处理的请求，它监控对战事件，跟踪对战状态并根据需要执行操作。根据您在游戏中管理游戏会话托管的方式，此服务可能会将游戏会话连接信息返回给玩家。此组件结合使用 AWS 开发工具包和 GameLift API 与 FlexMatch 服务进行通信。
- 对战放置服务（仅适用于作为独立服务的 FlexMatch）此组件可与您的现有游戏托管系统配合使用，以查找可用的托管资源并为对战启动新的游戏会话。–该组件必须获取对战结果并提取启动新游戏会话所需的信息，包括玩家 IDs、属性和对战中所有玩家的团队分配。

FlexMatch 对战过程

本主题介绍了一个基本对战方案以及各种游戏组件与 FlexMatch 服务之间的交互。

为玩家请求对战

使用您的游戏客户端的玩家单击“加入游戏”按钮。此操作会导致您的客户端对战服务向 FlexMatch 发送对战请求。该请求标识在完成请求时使用的 FlexMatch 对战构建器。该请求还包括您的自定义对战构建器所需的玩家信息，如技能级别、游戏首选项或地理延迟数据。您可以为一个玩家或多个玩家发出对战请求。

将请求添加到对战池

当 FlexMatch 收到对战请求时，它会生成对战票证并将其添加到对战构建器的票证池中。票证将保留在池中，直到匹配或达到最大时间限制。您的客户端对战服务会定期收到有关对战事件的通知，包括票证状态更改。

构建对战游戏

您的 FlexMatch 对战构建器将在其池的所有票证上持续运行以下进程：

1. 对战构建器按票证期限对池进行排序，然后开始从最旧的票证开始构建潜在对战游戏。
2. 对战构建器将第二个票证添加到潜在对战，并根据自定义对战规则评估结果。如果潜在对战游戏通过评估，票证的玩家将分配到团队。
3. 对战构建器按顺序添加下一个票证并重复评估过程。当填充所有玩家位置后，对战游戏已就绪。

大型对战游戏的对战游戏（41 对 200 位玩家）使用上述过程的修改版本，以便在合理的时间范围内构建对战游戏。对战构建器不会单独评估每个票证，而是将预排序票证池划分为潜在对战，然后根据您指定的玩家特征平衡每个对战游戏。例如，对战构建器可能会根据类似的低延迟位置对票证进行预排序，然后使用对战后平衡来确保团队按玩家技能平均匹配。

报告对战结果

找到可接受的对战游戏时，将更新所有匹配的票证，并为每个匹配的票证生成成功的对战事件。

- FlexMatch 作为独立服务：您的游戏收到对战事件成功对战。结果数据包括所有匹配的玩家及其团队分配的列表。如果您的对战请求包含玩家延迟信息，则结果还会为对战提供最佳地理位置建议。
- FlexMatch 与 GameLift 托管解决方案：匹配结果会自动传递到 GameLift 队列以进行游戏会话放置。对战构建器确定用于游戏会话放置的队列。

为对战游戏启动游戏会话

成功完成建议的对战游戏后，将启动新的游戏会话。在为对战设置游戏会话时，您的游戏服务器必须能够使用对战结果数据，包括玩家 IDs 和团队分配。

- FlexMatch 作为独立服务：您的自定义对战放置服务从成功的对战事件中获取对战结果数据，并连接到您的现有游戏会话放置系统，以查找对战的可用托管资源。找到托管资源后，对战放置服务将与现有托管系统协作以启动新游戏会话并获取连接信息。
- FlexMatch 与 GameLift 托管解决方案：游戏会话队列查找对战游戏的最佳可用游戏服务器。根据队列的配置方式，它将尝试使用成本最低的资源放置游戏会话，并且玩家将遇到低延迟（如果提供了玩家延迟数据）。成功放置游戏会话后，GameLift 服务将提示游戏服务器启动新游戏会话，并传递对战结果和其他可选的游戏数据。

将玩家连接到对战游戏

启动游戏会话后，玩家将连接到会话，声明其团队分配，然后开始游戏。

- FlexMatch 作为独立服务：您的游戏使用现有游戏会话管理系统向玩家提供连接信息。
- FlexMatch 与 GameLift 托管解决方案：在成功放置游戏会话后，FlexMatch 将使用游戏会话连接信息和玩家会话 ID 更新所有匹配的票证。

设置 FlexMatch

GameLift FlexMatch 是一项 AWS 服务，您必须拥有 AWS 账户才能使用此服务。创建 AWS 账户是免费的。有关如何使用 AWS 账户的更多信息，请参阅 AWS 入门。

如果您将 FlexMatch 与其他 GameLift 解决方案结合使用，请参阅以下主题：

- 设置 GameLift 托管和 的访问权限 实时服务器
- 使用 Amazon EC2 在 上设置托管的访问权限 GameLift FleetIQ

为 GameLift 设置您的账户

1. 获取账户。 打开 Amazon Web Services，然后选择 Sign In to the Console (登录到控制台)。按照提示创建新账户或登录现有账户。
2. 设置 管理用户组。 打开 AWS Identity and Access Management （IAM） 服务控制台，并按照步骤创建或更新用户或用户组。IAM 管理对您的 AWS 服务和资源的访问。必须向使用 FlexMatch 控制台或通过调用 GameLift GameLift 来访问您的 APIs 资源的每个人授予显式访问权限。有关使用控制台（或 AWS CLI 或其他工具）设置用户组的详细说明，请参阅创建 IAM 用户。
3. 将权限策略附加到您的 用户或用户组。对 AWS 服务和资源的访问通过将 IAM 策略附加到用户或用户组来管理。权限策略指定用户必须有权访问的一组 AWS 服务和操作。

   对于 GameLift，您必须创建自定义权限策略并将其附加到每个用户或用户组。策略是一个 JSON 文档。使用以下示例创建您的策略。

以下示例说明了对所有 GameLift 资源和操作具有管理权限的内联权限策略。您可以选择通过仅指定特定于 FlexMatch 的项目来限制访问。

```
{
"Version": "2012-10-17",
"Statement":
  {
    "Effect": "Allow",
    "Action": "gamelift:*",
    "Resource": "*"
  }
}
```

开始使用 FlexMatch

使用本节中的资源可帮助您开始使用 FlexMatch 构建对战系统。

主题

GameLift 用于单独对战的 FlexMatch 集成

本主题概述了将 FlexMatch 作为独立对战服务实施的完整集成过程。如果您的多人游戏是使用对等连接、自定义本地配置的硬件或其他云计算基元托管的，则使用此过程。此过程还可与 GameLift FleetIQ 一起使用，后者是适用于托管在 Amazon EC2 上的游戏的托管优化解决方案。如果您使用 GameLift 托管托管托管托管托管（包括 实时服务器）来托管游戏，请参阅FlexMatch 与 GameLift 托管的集成 (p. 8)。

在开始集成之前，您必须拥有 AWS 账户并设置 GameLift 服务的访问权限。有关详细信息，请参阅 设置 FlexMatch (p. 6)。与创建和管理 GameLift FlexMatch 对战构建器和规则集相关的所有关键任务都可以使用 Amazon GameLift 控制台完成，但您可能还需要 获取并安装 AWS Command Line Interface 工具。。

1. 创建 FlexMatch 对战规则集。 您的自定义规则集提供有关如何构建对战的完整说明。在其中，您可以定义每个团队的结构和规模。您还需要提供对战必须满足才有效的一组要求，FlexMatch 使用该要求在对战中包括或排除玩家。这些要求可能适用于单个玩家。您还可以自定义规则集中的 FlexMatch 算法，例如，与最多 200 位玩家构建大型对战游戏。请参阅这些主题：

   - 构建 FlexMatch 规则集 (p. 13)
   - FlexMatch 规则集示例 (p. 22)

2. 设置对战事件的通知。 使用通知跟踪 FlexMatch 对战活动，包括待处理对战请求的状态。这是用于提供建议的匹配项结果的机制。由于对战请求是异步的，您需要通过某种方式跟踪请求状态。使用通知是此项的首选选项。请参阅这些主题：

   - 设置 FlexMatch 事件通知 (p. 34)
   - FlexMatch 对战事件 (p. 58)

3. 设置 FlexMatch 对战配置。 此组件也称为对战构建器，它接收对战请求并对其进行处理。您可以通过指定规则集、通知目标和最长等待时间来配置对战构建器。您还可以启用可选功能。请参阅这些主题：

   - 设计 FlexMatch 对战构建器 (p. 9)
   - 创建对战配置 (p. 11)

4. 构建客户端对战服务。 创建或扩展具有生成对战请求并将其发送到 FlexMatch 的功能的游戏客户端服务。要构建对战请求，此组件必须具有相应的机制来获取对战规则集所需的玩家数据，以及（可选）区域延迟信息。它还必须具有为每个请求创建和分配唯一票证 IDs 的方法。您还可以选择构建要求玩家选择加入建议的对战游戏的玩家接受工作流程。请参阅以下主题：

   - 将 FlexMatch 添加到游戏客户端 (p. 36)

5. 构建对战放置服务。创建与您现有游戏托管系统配合使用的机制，以查找可用的托管资源并为对战启动新游戏会话。此组件必须能够检索 FlexMatch 对战游戏结果，并使用该信息将对战游戏与可用游戏服务器匹配，并指导它为对战设置新的游戏会话。您可能还希望实施一个工作流程以发出对战回填请求，该请求使用对战来填充已在运行的已对战游戏会话中的开放位置。

FlexMatch 与 GameLift 托管的集成

FlexMatch 适用于托管 GameLift，用于托管自定义游戏服务器和 实时服务器。要为您的游戏添加 FlexMatch 对战，请完成以下任务。

- 设置对战构建器。 对战构建器会接收玩家的对战请求并进行处理。它根据一组定义的规则将玩家分组，并为每个成功的对战游戏创建新的游戏会话和玩家会话。请按照以下步骤设置对战构建器：
  - 创建规则集。 规则集可以让对战构建器了解如何构建有效的对战游戏。它指定团队结构，并指定如何评估玩家是否参加对战游戏。请参阅这些主题：
    - 构建 FlexMatch 规则集 (p. 13)
    - FlexMatch 规则集示例 (p. 22)
  - 创建游戏会话队列。 队列查找每个对战游戏的最佳区域，并在该区域中创建新的游戏会话。使用现有队列或者为对战创建一个新队列。请参阅以下主题：
    - 创建队列
  - 设置通知 (可选)。 由于对战请求是异步的，您需要通过某种方式跟踪请求状态。通知是首选选项。请参阅以下主题：
    - 设置 FlexMatch 事件通知 (p. 34)
  - 配置一个对战构建器。 一旦您拥有了规则集、队列和通知目标，即可为您的对战构建器创建配置。请参阅这些主题：
    - 设计 FlexMatch 对战构建器 (p. 9)
    - 创建对战配置 (p. 11)
- 将 FlexMatch 集成到您的游戏客户端服务中。 向游戏客户端服务添加功能来启动包含对战的新游戏会话。对战请求指定要使用的对战构建器并为对战提供必要的玩家数据。请参阅以下主题：
  - 将 FlexMatch 添加到游戏客户端 (p. 36)
- 将 FlexMatch 集成到您的游戏服务器。 向游戏服务器添加功能来启动通过对战创建的游戏会话。此类游戏会话的请求包括对战特定信息，其中包括玩家和团队分配。在为对战构建游戏会话时，游戏服务器需要访问和使用这些信息。请参阅以下主题：
  - 将 FlexMatch 添加到 GameLift 托管的游戏服务器 (p. 40)
- 设置 FlexMatch 回填（可选）。 请求更多玩家匹配来填充现有游戏中空闲的玩家位置。您可以打开自动回填来让 GameLift 管理回填请求。或者，您可以通过向游戏客户端服务或游戏服务器添加功能以发起对战回填请求来手动管理回填。请参阅以下主题：
  - 使用 FlexMatch 回填现有游戏 (p. 41)

    Note

    FlexMatch 回填当前对使用 实时服务器 的游戏不可用。

构建 GameLift FlexMatch 对战构建器

FlexMatch 对战构建器进程可用于构建对战游戏。它可以管理接收的对战请求池、组建参加对战游戏的团队、处理和选择玩家以找到最佳玩家组，以及启动为对战游戏放置和启动游戏会话的进程。本主题介绍对战构建器的主要方面以及如何为您的游戏配置一个自定义对战构建器。

有关 FlexMatch 对战构建器如何处理它收到的对战请求的详细说明，请参阅 FlexMatch 对战过程 (p. 5)。

主题

设计 FlexMatch 对战构建器

本主题提供有关如何设计适合您的游戏的对战构建器的指导。

配置基本对战构建器

对战构建器至少需要以下元素：

- 规则集可确定对战团队的规模和范围并定义用于评估玩家是否参加对战的规则集。每个对战构建器均配置为使用一个规则集。请参阅 构建 FlexMatch 规则集 (p. 13) 和 FlexMatch 规则集示例 (p. 22)。
- 通知目标接收所有对战事件通知。您需要设置一个 Amazon Simple Notification Service （SNS） 主题，然后将主题 ID 添加到对战构建器。有关设置通知的更多信息，请参阅设置 FlexMatch 事件通知 (p. 34)。
- 请求超时可确定对战请求留在请求池中以及被评估为潜在对战游戏的时长。一旦请求超时，则无法进行对战，并将从池中删除。
- 将 FlexMatch 与 GameLift 托管托管托管托管结合使用时，游戏会话队列查找用于托管对战游戏会话的最佳可用资源，并启动新的游戏会话。每个队列配置了一个 AWS 区域和资源类型 （包括 Spot 或按需实例） 列表，用于确定游戏会话的放置位置。有关队列的更多信息，请参阅使用多区域队列。

为对战构建器选择一个 AWS 区域

确定您希望对战活动发生的位置，并在该区域中创建对战构建器（对战配置和规则集）。对战构建器的所有请求都将发送到该处票证池，在该处，将对可行对战游戏进行排序和评估。在进行对战游戏后，玩家可以定向到支持您的托管解决方案的任何位置的游戏会话。

为对战构建器选择 AWS 区域时，请考虑位置对其性能的影响以及如何优化目标玩家的对战体验。我们建议使用以下最佳实践：

- 将对战构建器放置在靠近玩家的 AWS 区域中 — 和发送 FlexMatch 对战请求的客户端服务。这种方法可减少对战请求工作流的延迟影响，并使其更高效。

- 如果您的游戏面向全球受众，请考虑在多个区域中创建对战构建器，并将对战请求路由到最靠近玩家的对战构建器。除了提高效率外，这会导致票证池与地理接近的玩家一起形成，这改进了对战构建器根据延迟要求匹配玩家的能力。
- 将 FlexMatch 与 GameLift 托管托管托管托管结合使用时，请将您的对战构建器和它使用的游戏会话队列放在同一个 AWS 区域中。这有助于最大程度地减少对战构建器和队列之间的通信延迟。

资源 FlexMatchMatchmakingConfiguration 和 MatchmakingRuleSet 可放置在以下 支持的 AWS 区域中：GameLift美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、欧洲中部（法兰克福）、欧洲西部（爱尔兰）、亚太地区东部（悉尼）和亚太地区亚太地区亚太地区太平洋（首尔和东京）。

添加可选元素

除了这些最低要求，您还可以为对战构建器配置以下附加选项。如果您将 FlexMatch 与 GameLift 托管解决方案结合使用，则会内置许多功能。如果您使用 FlexMatch 作为独立的对战服务，您可能需要在系统中构建这些功能。

玩家接受

您可以将对战构建器配置为要求为对战游戏选择的所有玩家必须接受参与游戏。如果您的系统需要接受，则必须给所有玩家提供接受或拒绝建议的对战游戏的选项。对战游戏必须收到建议对战游戏的所有玩家的接受信息，才能完成。如果任何玩家拒绝或未接受对战游戏，则会丢弃建议的对战游戏，并处理票证如下所示。票证中所有玩家接受对战游戏的票证将返回到对战池以继续处理。如果票证中至少有一个玩家拒绝了对战游戏或未能响应，则会置于失败状态，并且不再进行处理。玩家接受需要具有时间限制；所有玩家都必须在时间限制内接受建议的对战游戏才能继续对战。

回填模式

使用 FlexMatch 回填来在游戏会话的整个生命期让游戏会话始终有良好匹配的新玩家。在处理回填请求时，FlexMatch 使用与匹配原始玩家相同的对战构建器。您可以自定义回填票证的优先级以及新对战游戏的票证，从而将回填票证放入线的前端或末尾。这意味着，当新玩家进入对战池时，他们被放到现有游戏中的可能性高于或低于新形成游戏中的可能性。

无论游戏将 FlexMatch 与托管 GameLift 托管还是与其他托管解决方案一起使用，手动回填都可用。手动回填让您能够灵活地决定何时触发回填请求。例如，您可能希望仅在游戏的某些阶段或仅存在某些条件时添加新玩家。

自动回填仅适用于使用托管 GameLift 托管的游戏。启用此功能后，如果游戏会话从开放的玩家位置开始，GameLift 将开始自动生成它的回填请求。此功能允许您设置对战，以便为新游戏启动最少数量的玩家，然后在新玩家进入对战池时快速填充。您可以在游戏会话生命周期内随时关闭自动回填。

游戏属性

对于将 FlexMatch 与 GameLift 托管托管托管托管结合使用的游戏，您可以提供其他信息，以便在请求新的游戏会话时传递到游戏服务器。这是一种非常有用的方法，用于传递要为创建的对战游戏类型启动游戏会话所需的游戏模式配置。对战构建器创建的对战游戏的所有游戏会话都将收到一组相同的游戏属性。您可以通过创建不同的对战配置来改变游戏属性信息。

预留玩家位置

您可以指定为每个对战游戏预留的特定玩家位置，然后在日后占用这些位置。这可以通过配置对战配置的“额外玩家数量”属性完成。

自定义事件数据

使用此属性可在对战构建器的所有对战相关事件中包含一组自定义信息。此功能可用于跟踪您游戏独有的特定活动，包括跟踪对战构建器的性能。

创建对战配置

要设置 FlexMatch 对战构建器来处理对战请求，请创建对战配置。使用 GameLift 控制台或 AWS Command Line Interface (AWS CLI)。有关设计对战构建器的更多详细信息，请参阅the section called “设计对战构建器” (p. 9)。

为 GameLift 托管创建对战构建器

在创建对战配置之前，您必须创建规则集和 GameLift 游戏会话队列以用于对战构建器。

Console

1. 在 GameLiftgamelift/homehttps://console.amazonaws.cn/ 上打开 控制台。
2. 切换到要放置对战构建器的区域。有关支持 为对战构建器选择一个 AWS 区域 (p. 9) 对战配置的区域的列表，请参阅FlexMatch。
3. 在主菜单中，选择 Create matchmaking configuration。填充对战配置详细信息。
   - 名称 – 创建有意义的对战构建器名称，以便您可以轻松地在列表和指标中识别它。对战构建器名称在区域中必须唯一。对战请求会使用其名称和区域标识要使用的对战构建器。
   - 描述 – (可选) 添加对战构建器的描述。该描述仅用于标识；并不用于对战过程。
   - 请求超时 输入对战构建器为每个请求完成对战游戏的最长时间 (以秒为单位)。–超过该时间的对战请求都将终止。
   - FlexMatch mode – 选择“with queue”选项。这表示使用的是 GameLift 托管托管托管，并提示 FlexMatch 将成功的对战游戏传递到指定的游戏会话队列。
   - 队列 – 选择要用于该对战构建器的游戏会话队列。要查找队列，首先选择配置队列时所在的区域。然后，从该区域的可用队列列表中选择需要的队列。
   - 需要接受 – (可选) 指定是否要求建议对战游戏中的每个玩家主动接受参与对战游戏。如果选择是，表示您希望对战构建器在等待玩家接受多长时间之后取消对战游戏。
   - 规则集名称 – 从列表中选择用于此对战构建器的规则集。该列表包含在当前区域中已创建的所有规则集。
   - 通知目标 – (可选) 输入用于接收对战事件通知的 SNS 主题的 ARN。如果您尚未设置，可以在以后通过编辑对战配置来添加此信息。请参阅 设置 FlexMatch 事件通知 (p. 34)
   - 其他玩家 – (可选) 指定每个新对战游戏中仍然未占用的玩家位置数量。将来玩家可以占用这些位置。
   - 自定义事件数据 – (可选) 指定要与该对战构建器关联的、事件消息中的任何数据。该数据包含在与对战构建器关联的每个事件中。
   - 游戏数据 (可选) 提供要传输到每个新游戏会话 (通过使用此对战配置进行的对战游戏) 的其他信息。–您可以提供单个字符串值 (使用 Game session data (游戏会话数据) 字段) 或一组键值对 (使用 Game properties (游戏属性) 字段) 形式的信息。
4. 完成配置对战构建器后，单击 Create。如果创建成功，则对战构建器会立即准备好接受对战请求。

AWS CLI

要使用 AWS CLI 创建对战配置，请打开命令行窗口，然后使用 `create-matchmaking-configuration` 命令定义一个新对战构建器。请参阅 AWS CLI 命令参考中对此命令的完整介绍。获取并安装 AWS Command Line Interface 工具。

此示例创建一个新的对战配置，要求玩家接受并启用自动回填。它还为其他玩家预留两个位置以便以后添加，并提供一些游戏会话数据。

```
$ aws gamelift create-matchmaking-configuration
```

```
--name "SampleMatchamker123"
--description "The sample test matchmaker with acceptance"
--flexmatch-mode "WITH_QUEUE"
--game-session-queue-arns "arn:aws-cn:gamelift:us-west-2:111122223333:gamesessionqueue/
My_Game_Session_Queue_One"
--rule-set-name "My_Rule_Set_One"
--request-timeout-seconds "120"
--acceptance-required "true"
--acceptance-timeout-seconds "30"
--backfill-mode "AUTOMATIC"
--notification-target "arn:aws-cn:sns:us-west-2:111122223333:My_Matchmaking_SNS_Topic"
--additional-player-count "2"
--game-session-data "key=map,value=winter444"
```

可复制版本：

```
aws gamelift create-matchmaking-configuration --name "SampleMatchamker123"
 --description "The sample test matchmaker with acceptance" --flexmatch-
mode "WITH_QUEUE" --game-session-queue-arns "arn:aws-cn:gamelift:us-
west-2:111122223333:gamesessionqueue/My_Game_Session_Queue_One" --rule-set-name
 "My_Rule_Set_One" --request-timeout-seconds "120" --acceptance-required "true" --
acceptance-timeout-seconds "30" --backfill-mode "AUTOMATIC" --notification-target
 "arn:aws-cn:sns:us-west-2:111122223333:My_Matchmaking_SNS_Topic" --additional-player-
count "2" --game-session-data "key=map,value=winter444"
```

如果对战配置创建请求成功，GameLift 会返回一个 MatchmakingConfiguration 对象以及为对战构建器请求的设置。对战构建器会立即准备好接受对战请求。

为独立 FlexMatch 创建对战构建器

在创建对战配置之前，您必须创建一个规则集以用于对战构建器。

Console

1. 在 GameLiftgamelift/homehttps://console.amazonaws.cn/ 上打开 控制台。
2. 切换到要放置对战构建器的区域。有关支持 为对战构建器选择一个 AWS 区域 (p. 9) 对战配置的区域的列表，请参阅FlexMatch。
3. 在主菜单中，选择 Create matchmaking configuration。填充对战配置详细信息。
   - 名称 – 创建有意义的对战构建器名称，以便您可以轻松地在列表和指标中识别它。对战构建器名称在区域中必须唯一。对战请求会使用其名称和区域标识要使用的对战构建器。
   - 描述 – (可选) 添加对战构建器的描述。该描述仅用于标识；并不用于对战过程。
   - 请求超时 – 键入对战构建器针对每个请求完成对战游戏的最长时间 (以秒为单位)。超过该时间的对战请求都将终止。
   - FlexMatch mode – 选择“standalone”选项。这指示游戏具有自定义机制，用于在非 GameLift 托管解决方案上启动新游戏会话。
   - 需要接受 – (可选) 指定是否要求建议对战游戏中的每个玩家主动接受参与对战游戏。如果选择是，则指示您希望对战构建器在取消对战游戏之前等待玩家接受的时间。
   - 规则集名称 – 选择要用于该对战构建器的规则集。该列表包含在当前区域中已创建的所有规则集。
   - 通知目标 – (可选) 键入用于接收对战事件通知的 SNS 主题的 ARN。如果您尚未设置，可以在以后通过编辑对战配置来添加此信息。请参阅 设置 FlexMatch 事件通知 (p. 34)。
   - 自定义事件数据 – (可选) 指定要与该对战构建器关联的、事件消息中的任何数据。该数据包含在与对战构建器关联的每个事件中。
4. 完成配置对战构建器后，单击 Create。如果创建成功，则对战构建器会立即准备好接受对战请求。

AWS CLI

要使用 AWS CLI 创建对战配置，请打开命令行窗口，然后使用 `create-matchmaking-configuration` 命令定义一个新对战构建器。请参阅 AWS CLI 命令参考中对此命令的完整介绍。获取并安装 AWS Command Line Interface 工具。

此示例为需要玩家接受的独立对战构建器创建新的对战配置。

```
$ aws gamelift create-matchmaking-configuration
--name "SampleMatchamker123"
--description "The sample test matchmaker with acceptance"
--flexmatch-mode "STANDALONE"
--rule-set-name "My_Rule_Set_One"
--request-timeout-seconds "120"
--acceptance-required "true"
--acceptance-timeout-seconds "30"
--backfill-mode "MANUAL"
--notification-target "arn:aws-cn:sns:us-west-2:111122223333:My_Matchmaking_SNS_Topic"
```

可复制版本：

```
aws gamelift create-matchmaking-configuration --name "SampleMatchamker123" --
description "The sample test matchmaker with acceptance" --flexmatch-mode "STANDALONE"
 --rule-set-name "My_Rule_Set_One" --request-timeout-seconds "120" --acceptance-
required "true" --acceptance-timeout-seconds "30" --backfill-mode "MANUAL" --
notification-target "arn:aws-cn:sns:us-west-2:111122223333:My_Matchmaking_SNS_Topic"
```

如果对战配置创建请求成功，GameLift 会返回一个 MatchmakingConfiguration 对象，其中包含您为对战构建器请求的设置。对战构建器会立即准备好接受对战请求。

构建 FlexMatch 规则集

每个 FlexMatch 对战构建器必须拥有一个规则集。规则集可确定对战游戏的两个关键元素：游戏团队的结构和规模，以及如何为玩家分组才能呈现可能的最佳对战游戏。

例如，规则集可能描述如下所示的匹配项：创建包含两个团队的对战游戏，每个团队有五个玩家，一个团队是防守者，另一个团队是进进进进者。团队可以有新手玩家和经验丰富的玩家，但两个团队的平均技能必须在 10 点以内。如果在 30 秒后未能进行匹配，则逐渐放宽技能要求。

本部分的主题介绍如何设计和构建对战规则集。在创建规则集时，您可以使用 Amazon GameLift 控制台或 AWS CLI。

主题

设计 FlexMatch 规则集

在最基本的层面上，对战规则集有两个作用：安排对战游戏的团队结构和规模，以及告知对战构建器如何评估玩家以查找可能的最佳匹配。但是，它能够做的比这些要多得多。例如，您还可以通过规则集解决地址匹配问题，如：

- 为您的游戏优化匹配算法。
- 为大型对战（> 40 位玩家）触发特殊的匹配处理。
- 强制实施最低玩家延迟要求以保护游戏质量。
- 如果无法进行匹配，则逐步放宽团队要求或匹配规则。
- 为包含多个玩家（各方聚合）的对战请求定义特殊处理。

本主题介绍规则集的基本结构以及如何将它们用于最多 40 位玩家的对战游戏。超过 40 位玩家的对战游戏需要不同的规则集结构，因为 FlexMatch 使用简化的算法来快速匹配大量玩家。在 设计 FlexMatch 大型对战规则集 (p. 18) 中了解有关构建大型对战游戏的更多信息。

相关主题

- 创建对战规则集 (p. 20)
- FlexMatch 规则集示例 (p. 22)
- FlexMatch 规则语言 (p. 54)
- FlexMatch 规则集架构 (p. 48)

定义规则集组件

所有规则集包含部分或全部组件，如FlexMatch 规则集架构 (p. 48)中的一般规则集架构中所述。本主题介绍每个规则集组件的设计问题：

- 描述规则集 (p. 14)
- 自定义匹配算法 (p. 14)
- 声明玩家属性 (p. 17)
- 定义对战团队 (p. 17)
- 设置玩家匹配规则 (p. 17)
- 允许要求随时间推移放宽 (p. 18)

需要规则集来构建超过 40 位玩家的对战游戏？了解如何设计 FlexMatch 大型对战规则集 (p. 18)。

描述规则集

提供规则集的详细信息。

- name（可选）– 这是规则集语法中的描述性标签，不能供 Amazon GameLift 以任何有意义的方式使用。不要将此值与规则集名称混淆，规则集名称在创建规则集时与规则集语法一同设置。
- ruleLanguageVersion（必需） 这是用于创建 – 规则的属性表达式语言的版本。FlexMatch此值必须等于“1.0”。

自定义匹配算法

您可以选择修改默认匹配算法的某些元素。从设计上讲，默认算法针对大多数游戏进行了优化，以便快速高效地处理对战请求，从而使玩家可在最短的等待时间内加入到可接受的对战游戏。您可以自定义算法并调整对战优先级，以更好地满足您的游戏需求。

默认 FlexMatch 对战过程如下所示：

1. 所有开放对战票证和回填票证都放置在票证池中。
2. 票证随机分组为一个或多个批次以进行匹配。仅当票证池变得太大以获得最佳匹配时，才会形成多个批次。
3. 每个批次按票证期限进行排序。

4. 从最旧的票证开始，将对批处理中的所有票证进行评估以找到可接受的匹配项。

以下建议的自定义会影响对战过程的不同阶段。要自定义匹配算法，请将 `algorithm` 组件添加到规则集架构中。有关完整的参考信息，请参阅FlexMatch 规则集架构 (p. 48)。

添加批处理前排序

您可以将 FlexMatch 配置为在形成批次之前对票证池进行排序。这种类型的自定义与往往具有非常大的票证池的游戏最有效。前批处理排序可用于帮助加快对战过程，它还有助于在某些特征中构建具有更高玩家性能的对战游戏。

在算法属性 `batchingPreference` 中定义了预批量排序方法。 默认设置为“随机”，这指示没有发生预排序。

用于自定义预批量排序的选项包括：

- 按玩家属性排序。 提供要对票证池进行预排序的玩家属性列表。此自定义会导致 FlexMatch 在排序的属性中创建更加一致的批处理。例如，如果您按玩家技能对票证池进行预先排序，则具有相似技能级别的票证往往会被批处理在一起。如果您的规则集还包含基于玩家技能的对战规则，则预批量排序可以显著提升对战效率。

  要启用此自定义，请将算法属性 `batchingPreference` 设置为“sorted”，并将属性 `sortByAttributes` 设置为玩家属性的列表。列表中的每个属性在规则集的 `playerAttributes` 组件中均有声明。

  在以下示例中，FlexMatch 首先根据玩家的首选游戏地图，然后按玩家技能对票证池进行排序。生成的批次更有可能包含希望使用相同地图的技能类似的玩家。

```
"algorithm": {
    "batchingPreference": "sorted",
    "sortByAttributes": ["map", "player_skill"],
    "strategy": "exhaustiveSearch"
},
```

- 按延迟进行排序。 在下列选项之一之间进行优先选择：（1）将玩家置于可用延迟最低的对战中，或（2）快速将玩家置于可接受延迟的对战中。此自定义适合用于组成大型对战游戏的规则集（超过 40 位玩家）；算法属性 `strategy` 必须设置为“balanced”，这会显著限制规则语句的可用类型。请参阅 设计 FlexMatch 大型对战规则集 (p. 18)。

  此自定义会导致 FlexMatch 根据报告的延迟数据通过以下方式之一对票证进行预排序：

  - 让玩家进入延迟最低的区域。票证池按玩家报告其最低延迟值的区域预先排序。这会导致 FlexMatch 批处理在同一区域中报告低延迟的票证，从而倾向于将玩家匹配到其最快区域并改善整体游戏体验。它还会减少每个批次中的票数，因此对战游戏可能需要更长时间才能完成。要启用此自定义，请将算法属性 `batchingPreference` 设置为值“fastestRegion”，如以下示例所示。

```
"algorithm": {
    "batchingPreference": "fastestRegion",
    "strategy": "balanced"
},
```

  - 快速让玩家进入可接受的延迟匹配。票证轮询按玩家在其中报告任何可接受的延迟值的区域预先排序。此方法往往会形成更少的批次，每个批次包含大量在同一区域中具有可接受延迟的票证。每个批次中的票证越多，查找足够的可接受对战往往更轻松快捷。要启用此自定义，请将属性 `batchingPreference` 设置为值“largestPopulation”，如以下示例所示。（此方法是使用平衡策略的规则集的默认行为。）

```
"algorithm": {
    "batchingPreference": "largestPopulation",
    "strategy": "balanced"
```

```
},
```

确定回填票证的优先级

如果您的游戏实施自动回填或手动回填，您可以自定义 FlexMatch 如何根据请求类型（新对战请求或回填请求）处理对战票证。默认情况下，两种类型的请求都得到相同处理。此自定义会导致 FlexMatch 先尝试填充回填票证，或者仅在无法进行新的对战时尝试填充。它还可能会影响托管资源的使用 – ，方式是将玩家打包到较少的游戏会话或者创建更多部分填充的游戏会话。

回填优先级会影响在对票证进行批处理后的处理方式；它不会影响票证批处理过程。您可以根据需要实施预批量排序和回填优先级。仅将回填优先级用于使用详尽搜索策略的规则集。

要更改回填票证的优先级，请设置属性 `backfillPriority`，如下所示：

- 首先匹配回填票证。此选项提示 FlexMatch 在创建新对战之前评估并尝试完成回填票证。这意味着，传入玩家被放入现有游戏中的可能性更高。将属性 `backfillPriority` 设置为“high”。

  如果您的游戏使用的是自动回填，请启用此自定义项作为最佳实践。自动回填最常用于游戏，具有较短的游戏会话时间范围和高度玩家周转。自动回填有助于这些游戏快速形成最小可行的对战游戏并开始，即使 FlexMatch 搜索更多玩家来填充空闲位置。首先尝试填充回填票证有助于优化此方法。

```
"algorithm": {
    "backfillPriority": "high",
    "strategy": "exhaustiveSearch"
},
```

- 匹配回填票证最后。此选项将提示 FlexMatch 忽略回填票证，直到所有其他票证都已评估。这意味着，仅在传入玩家无法匹配到新游戏时，才将他们回填到现有游戏中。将属性 `backfillPriority` 设置为“low”。

  当您想要使用回填作为最后一项通道选项来让玩家进入游戏时（例如，当玩家太少而无法组成新对战时），此选项很有用。取消回填票证的优先级不应与自动回填一起使用。

```
"algorithm": {
    "backfillPriority": "low",
    "strategy": "exhaustiveSearch"
},
```

首选具有扩展的旧票证

自定义 FlexMatch 如何应用扩展规则，以便在对战游戏难以完成时放宽匹配条件。GameLift 在部分完成的对战游戏中的票证达到特定期限时应用扩展规则。票证的创建时间戳确定何时应用规则；默认情况下，FlexMatch 跟踪最近匹配的票证的时间戳。

要在应用扩展规则时进行更改，请设置属性 `expansionAgeSelection`，如下所示：

- 根据最新票证进行扩展。此选项根据添加到潜在对战游戏的最新票证应用扩展规则。每次匹配新票证时，都会重置时钟。使用此选项，应用扩展会更加困难；生成的对战游戏往往会获得更高的质量，但玩家的等待时间可能会更长，对战请求在完成前可能会超时。将属性 `expansionAgeSelection` 设置为“新”（这是默认值）。
- 根据最旧的票证进行扩展。此选项根据潜在对战游戏中最早的票证（通常是（但并非总是）添加到对战游戏的第一个票证）应用扩展规则。使用此选项，扩展的应用速度往往更快，这可以缩短最早的匹配玩家的等待时间，同时降低所有玩家的匹配质量。将属性 `expansionAgeSelection` 设置为“oldest”。

```
"algorithm": {
    "expansionAgeSelection": "oldest",
```

```
    "strategy": "exhaustiveSearch"
},
```

声明玩家属性

规则可能会根据各个玩家的特性选择对战游戏的玩家。如果您创建了依赖于玩家属性的规则，则必须在本部分中声明。声明的玩家属性的值应该包含在使用此规则集发送对战构建器的每个对战请求中。

您可能还需要将某些玩家属性传递到游戏会话中，即使规则集在玩家评估期间不使用这些属性。例如，您可以传递选择的玩家角色。要执行此操作，请在此处声明您的玩家属性，并在对战请求中包含每个玩家的属性值。

在声明玩家属性时，请包含以下信息：

- name（必需）– 此值在规则集中必须唯一。
- type（必需）– 这是属性值的数据类型。有效数据类型为数字、字符串或字符串映射。
- default（可选）– 输入在未为玩家提供值时要使用的默认值。如果没有声明默认值，玩家也没有提供值，则无法匹配该玩家。

定义对战团队

描述对战游戏的团队的结构和规模。每个对战游戏必须至少有一个团队，您可以根据需要定义任意数量的团队。您的团队的玩家数量可以相同，也可以不同。例如，您可以定义有一个玩家的怪物团队和有 10 个玩家的猎人团队。

FlexMatch 根据规则集如何定义团队规模将对战请求处理为小型对战或大型对战。最多 40 位玩家的潜在对战游戏属于小型对战，而超过 40 位玩家的对战游戏则属于大型对战。要确定规则集的潜在对战游戏规模，请添加规则集中定义的所有团队的 maxPlayer 设置。

- name（必需）– 为每个团队分配一个唯一名称。此名称在规则和扩展中使用，并在用于游戏会话的对战数据中引用。
- maxPlayers（必填） 指定可以分配给团队的玩家的最大数量。–
- minPlayers（必填） 指定要使对战成功必须分配给团队的玩家的最小数量。–
- quantity（可选）– 如果您希望 FlexMatch 根据此定义创建多个团队，请指定数量。FlexMatch 创建对战游戏时，会为这些团队提供指定名称并附加一个编号。例如，“Red-Team_1”、“Red-Team_2”、“Red-Team_3”等。

FlexMatch 始终会尝试将团队填充到最大玩家规模，但在创建团队时则会设置最小玩家规模允许的较少的玩家数量。如果您希望对战游戏的所有团队都具有相同规模，可以创建相应的规则。有关“EqualTeamSizes”规则的示例，请参阅 FlexMatch 规则集示例 (p. 22)主题。

设置玩家匹配规则

创建一组定义如何评估玩家在对战游戏中的接受情况的规则语句。规则可以设置应用于单个玩家、团队或整个对战的要求。当 GameLift 处理对战请求时，它将从可用玩家池中最早的玩家开始，围绕该玩家构建对战。

- name（必需）– 这是用于在规则集中唯一标识规则的有意义的名称。规则名称也可在跟踪与此规则相关的活动的事件日志和指标中引用。
- description（可选）– 使用此元素可附加自由格式文本描述。此信息不可供对战构建器使用。
- type（必需）– 类型元素标识处理规则时要使用的操作。每个规则类型都需要一组额外属性。例如，若干规则类型需要一个引用值来衡量玩家的属性。有关有效的规则类型和属性的列表，请参阅 FlexMatch 规则语言 (p. 54)。
- 规则类型属性（可能是必需的）– 根据定义的规则类型，您可能需要设置某些规则属性。例如，对于距离和比较规则，必须指定要衡量的玩家属性。在 FlexMatch 规则语言 (p. 54) 中了解有关属性以及如何使用 FlexMatch 属性表达式语言的更多信息。

允许要求随时间推移放宽

扩展允许您在无法完成对战时随着时间推移放宽匹配条件。此功能可确保在不可能有完美匹配时，进行“现有最佳”匹配。您可以使用扩展来放宽玩家技能要求，提高可接受的玩家延迟级别，或减少团队中所需的最小玩家数量。通过利用扩展放宽规则，您可以逐渐扩展可接受对战游戏的玩家池。

当未完成匹配中最新票证的期限与扩展等待时间匹配时，将触发扩展。在对战游戏中添加新票证时，可能会重置扩展等待时间时钟。您可以在规则集的 `algorithm` 部分中自定义如何触发扩展。请记住， 中的扩展步骤等待时间是绝对的，它们不会相互复合。因此，如果您有两个步骤，一个步骤设置为 15，另一个步骤设置为 30，则第一个步骤将在第一个步骤后 15 秒触发。

下面是一个扩展的示例，该扩展会逐步提高对战游戏所需的最低技能级别。规则集使用名为“SkillDelta”的距离规则语句来要求对战游戏中的所有玩家都在 5 个技能级别以内。如果在 15 秒内没有进行新的匹配，则此扩展允许技能级别差异 10，然后 10 秒允许差异 20。

```
"expansions": [{
        "target": "rules[SkillDelta].maxDistance",
        "steps": [{
            "waitTimeSeconds": 15,
            "value": 10
        }, {
            "waitTimeSeconds": 25,
            "value": 20
        }]
    }]
```

启用自动回填的对战构建器不会太快地放宽玩家计数要求。新游戏会话需要几秒钟时间启动并开始自动回填。如果您的扩展步骤具有非常短的等待时间，FlexMatch 可能会在新启动的游戏会话发送回填请求之前创建大量部分填充的对战。更好的方法是仅在您的游戏开始自动回填后触发扩展。这有助于 FlexMatch 更快、更高效地让玩家进入游戏（新游戏或现有游戏）。扩展时间因您的团队构成而异，因此，预计会进行一些测试来找到最适合您的游戏的扩展策略。

设计 FlexMatch 大型对战规则集

如果您的规则集创建的对战允许 41 到 200 位玩家，则需要对规则集配置做出一些调整。这些调整会优化对战算法，使其可以构建可行的大型对战，同时确保玩家等待时间较短。因此，大型对战规则集将耗时的自定义规则替换为针对常见对战优先级进行了优化的标准解决方案。

下面介绍如何确定您是否需要针对大型对战优化您的规则集：

1. 对于规则集中定义的每个团队，获取 maxPlayer 的值。
2. 将所有 maxPlayer 值相加。如果总数超过 40，您有大型对战规则集。

要针对大型对战优化您的规则集，请按以下所示进行调整。请参阅 大型对战的规则集架构 (p. 50) 中的大型对战规则集架构以及 示例 7：创建大型对战 (p. 31) 中的规则集示例。

为大型对战自定义对战算法

将算法组件添加到规则集（如果尚不存在）。设置以下属性。

- `strategy`（必需）– 将 `strategy` 属性设置为“balanced”。此设置将触发 FlexMatch 进行其他对战后检查，以根据指定的玩家属性查找最佳团队平衡。使用 `balancedAttribute` 属性设置玩家属性。平衡策略可免除自定义规则构建平均对战团队的需求。
- `balancedAttribute`（必需）– 标识在对战中平衡团队时要使用的玩家属性。此属性必须具有数字数据类型（双精度或整数）。例如，如果您选择平衡玩家技能，FlexMatch 会尝试分配玩家，以便所有团队拥有尽可能均匀的技能级别。必须在规则集的玩家属性中声明平衡属性。

- `batchingPreference`（可选）– 选择您希望为玩家构建最低延迟的对战游戏的强调。此设置会影响在构建对战游戏之前对战游戏票证的排序方式。选项包括：
  - 最大群体。FlexMatch 允许对战池中所有具有可接受延迟值且在至少一个区域中共有的票证。因此，潜在票证池往往很大，从而可以更轻松地更快地填充匹配项。玩家可能会被放入具有可接受但并非始终最佳的延迟的游戏中。如果未设置此属性，则这是当 `strategy` 设置为“balanced”时的默认行为。
  - 最快的区域。FlexMatch 根据报告最低延迟值的位置对池中的所有票证进行预排序。因此，通常与在同一区域中报告低延迟的玩家形成对战游戏。同时，每个对战游戏的潜在票证池较小，这可能会增加填充对战游戏所需的时间。此外，由于对延迟设置了更高的优先级，对战游戏中的玩家在平衡属性方面可能会变化更大。

以下示例将匹配算法配置为按如下所示操作：（1）根据具有可接受的延迟值的区域对票证池进行预排序以便对票证进行分组；（2）为匹配设置已排序票证批次；（3）通过一个批次中的票证创建对战，平衡团队以平衡平均玩家技能。

```
"algorithm": {
    "strategy": "balanced",
    "balancedAttribute": "player_skill",
    "batchingPreference": "largestPopulation"
},
```

声明玩家属性

确保在规则集算法中声明用作平衡属性的玩家属性。该属性应包含每个玩家的对战请求中。您可以为玩家属性提供默认值，但在提供特定于玩家的值时，属性平衡效果最佳。

定义团队

定义团队规模和结构的过程与小型对战相同，但 FlexMatch 填充团队的方式不同。这会影响在只有部分填充时对战可能呈现的外观。您可能想要更改响应中的最小团队规模。

FlexMatch 在向团队分配玩家时使用以下规则。首先：查找尚未达到其最低玩家要求的团队。其次：在这些团队中，查找具有最多空闲位置的团队。

对于定义多个同等规模团队的对战，玩家将按顺序添加到每个团队，直到填满。因此，对战游戏的团队始终拥有几乎相等的玩家数量，即使对战游戏未填满。目前，还没有方法在大型对战中强制定义规模相同的团队。对于规模不对称的团队，过程稍微复杂一些。在这种情况下，玩家最初被分配给空闲位置最多的最大团队。随着空闲位置的数量在所有团队中更均匀地分配，玩家被排入更小的团队。

例如，假设您有一个包含三个团队的规则集。红色团队和蓝色团队均设置为 `maxPlayers=10`、`minPlayers=5`。绿色团队设置为 `maxPlayers=3`，`minPlayers=2`。填充序列如下：

1. 没有团队已达到 `minPlayers`。 红色团队和蓝色团队有 10 个空闲位置，绿色团队有 3 个。前 10 个玩家被分配（每个团队 5 个）到红色团队和蓝色团队。这两个团队现在已达到 `minPlayers`。
2. 绿色团队尚未达到 `minPlayers`。 接下来 2 个玩家被分配到绿色团队。绿色团队现已达到 `minPlayers`。
3. 所有团队现在位于 `minPlayers`，现在将根据空闲位置的数量分配其他玩家。红色团队和蓝色团队各有 5 个空闲位置，绿色团队有 1 个。接下来 8 个玩家将被分配（每个团队 4 个）到红色团队和蓝色团队。所有团队现在都有 1 个空闲位置。
4. 剩下的 3 个玩家位置分配（每个位置 1 个）到团队中，无特定顺序。

为大型对战设置延迟规则

大型对战的对战主要取决于平衡策略和延迟批处理优化。大多数自定义规则不可用。但是，您可以创建一个对玩家延迟设置硬性限制的规则。

要创建此规则，请使用属性为 `latency` 的 `maxLatency` 规则类型。 以下示例将最大玩家延迟设置为 200 毫秒：

```
"rules": [{
        "name": "player-latency",
        "type": "latency",
        "maxLatency": 200
    }],
```

放宽大型对战要求

在小型对战中，您可以使用扩展来在可能没有有效匹配时随着时间推移放宽匹配要求。对于大型对战，您可以选择放宽延迟规则或团队玩家计数。

如果您在大型对战中使用自动匹配回填，请不要太快地放宽团队玩家计数。FlexMatch 仅在游戏会话启动后开始生成回填请求，在对战创建后的前几秒钟可能不会开始。在这段时间内，FlexMatch 可以创建多个部分填充的新游戏会话，尤其是在玩家计数规则降低时。因此，您最后将有多于所需的更多游戏会话，而且玩家将在这些会话之间过于稀疏地分布。最佳做法是给予玩家计数扩展的第一个步骤更长的等待时间，长到足够让游戏会话启动。由于为大型对战的回填请求提供了更高的优先级，传入玩家将在新游戏启动前被放入现有游戏。您可能需要进行实验来找到最适合您的游戏的等待时间。

下面是一个在更长的初始等待时间内逐渐降低黄色团队的玩家计数的示例。请记住，规则集扩展中的等待时间是绝对的，不是复合的。因此，第一次扩展在第五秒时进行，第二次扩展在五秒以后，即第十秒时进行。

```
"expansions": [{
        "target": "teams[Yellow].minPlayers",
        "steps": [{
            "waitTimeSeconds": 5,
            "value": 8
        }, {
            "waitTimeSeconds": 10,
            "value": 5
        }]
    }]
```

创建对战规则集

为您的 FlexMatch 对战构建器管理对战规则集。使用 Amazon GameLift 控制台或 AWS 命令行界面（CLI）。在 the section called “FlexMatch 的工作原理” (p. 2) 中了解 FlexMatch 对战如何运行的更多信息。

对战规则集一旦创建就无法更改，因此我们建议先检查规则集语法，然后再创建规则集。控制台和 AWS CLI 均提供了验证选项。您可以拥有的规则集有最大数量限制，因此，最好将不使用的规则集删除。

相关主题

- 设计 FlexMatch 规则集 (p. 13)
- FlexMatch 规则集示例 (p. 22)
- FlexMatch 规则语言 (p. 54)

Console

创建规则集：

1. 在 Amazon GameLiftgamelift/https://console.amazonaws.cn/ 打开 控制台。
2. 切换到您希望放置规则集的区域。规则集必须在要使用对战配置的同一个区域中定义。
3. 从 Amazon GameLift 主菜单中，选择 Create matchmaking rule set (创建对战规则集)，然后填写规则集详细信息。

- 规则集名称 – 创建有意义的名称，以便您可以轻松地在列表以及在事件和指标中识别它。规则集名称在区域中必须唯一。对战配置按名称确定要使用的规则集。注意：这与规则集正文中的“name”字段不同，目前未使用该字段。
- 规则集 – 输入规则集正文的 JSON 文本。在 the section called “设计规则集” (p. 13) 中了解有关设计规则集的更多信息，或使用 FlexMatch 规则集示例 (p. 22) 中的其中一个示例规则集。

4. 由于规则集在创建之后无法编辑，因此先验证您的规则集是一种好做法。单击 Validate rule set 以验证您规则集正文的语法正确。
5. 在您完成配置对战构建器之后，单击 Create rule set。如果创建成功，对战构建器可以使用该规则集。

删除规则集：

1. 在对战规则集控制台页面上，选择一个规则集，然后单击 Delete rule set (删除规则集)。
2. 如果要删除的规则集目前正被某个对战配置使用，则会显示一条错误消息。在这种情况下，必须更改对战配置以使用其他规则集，然后才能删除该规则集。要了解一个规则集目前正被哪些对战配置使用，请单击该规则集名称，查看其详细信息页面。

AWS CLI

创建规则集：

- 要使用 AWS CLI 创建对战规则集，请打开一个命令行窗口，使用命令 `create-matchmaking-rule-set`（AWS CLI 命令参考）。获取并安装 AWS Command Line Interface 工具。

  此示例创建设置单个团队的简单对战规则集。请确保创建规则集所在的区域，与将引用该规则集的对战配置的区域相同。

```
$ aws gamelift create-matchmaking-rule-set
--name "SampleRuleSet123"
--rule-set-body '{"name":  "aliens_vs_cowboys",
     "ruleLanguageVersion":  "1.0",
     "teams": [{
         "name":  "cowboys",
         "maxPlayers":  8,
         "minPlayers":  4}]}'
```

  可复制版本：

```
aws gamelift create-matchmaking-rule-set --name "SampleRuleSet123" --rule-set-body
 '{"name": "aliens_vs_cowboys", "ruleLanguageVersion": "1.0", "teams": [{"name":
 "cowboys", "maxPlayers": 8, "minPlayers":  4}]}'
```

  如果创建请求成功，Amazon GameLift 将返回一个 MatchmakingRuleSet 对象，其中包含您指定的设置。新规则集现在可由对战构建器使用。

删除规则集：

- 要使用 AWS CLI 删除一个对战规则集，请打开一个命令行窗口，使用命令 `delete-matchmaking-rule-set`（AWS CLI 命令参考）。

如果要删除的规则集目前正被某个对战配置使用，则会显示一条错误消息。在这种情况下，必须更改对战配置以使用其他规则集，然后才能删除该规则集。要获取当前使用某个规则集的对战配置列表，请使用命令 `describe-matchmaking-configurations`（AWS CLI 命令参考）并指定该规则集名称。

此示例首先检查对战规则集的使用情况，然后删除该规则集。

```
$ aws gamelift describe-matchmaking-configurations
--rule-set-name "SampleRuleSet123"
--limit 10

$ aws gamelift delete-matchmaking-rule-set
--name  "SampleRuleSet123"
```

可复制版本：

```
aws gamelift describe-matchmaking-configurations --rule-set-name "SampleRuleSet123"
 --limit 10
```

```
aws gamelift delete-matchmaking-rule-set --name "SampleRuleSet123"
```

如果删除请求成功，则 Amazon GameLift 返回成功。

FlexMatch 规则集示例

FlexMatch, 规则集可以涵盖各种对战情况。以下示例符合 FlexMatch 配置结构和属性表达式语言。完整复制这些规则集或根据需要选择组件。

有关使用 FlexMatch 规则和规则集的更多信息，请参阅以下主题：

- 构建 FlexMatch 规则集 (p. 13)
- 设计 FlexMatch 规则集 (p. 13)
- FlexMatch 规则集架构 (p. 48)
- FlexMatch 规则语言 (p. 54)

> **Note**
>
> 当评估包含多个玩家的对战票证时，请求中的所有玩家都必须满足对战要求。

示例 1：创建两个对战玩家的团队

此示例说明如何按照以下说明设置两个势均力敌的玩家对战团队。

- 创建两个玩家团队。
  - 每个团队包含四到八个玩家。
  - 最终团队必须具有相同数量的玩家。
- 附上玩家的技能水平 (如果未提供，则默认为 10)。
- 选择与其他玩家技能水平相当的玩家。确保这两个团队的平均玩家技能在 10 点以内。
- 如果未能快速填充对战游戏，则放宽玩家技能要求以在合理的时间内完成对战游戏。
  - 5 秒之后，扩大搜索范围以允许创建平均玩家技能在 50 点以内的团队。
  - 15 秒之后，扩大搜索范围以允许创建平均玩家技能在 100 点以内的团队。

使用此规则集的说明：

- 此示例允许创建包含 4 到 8 位玩家的任何规模的团队 (但两个团队的规模必须相同)。对于在有效规模范围内的团队，对战构建器会尽量尝试匹配允许的最大玩家数量。
- `FairTeamSkill` 规则集可确保根据玩家技能构建对战团队。要对每个新的潜在玩家评估此规则，FlexMatch, 会暂时将玩家加入团队并计算平均值。如果规则失败，则不会将潜在玩家添加到对战游戏。
- 由于这两个团队具有相同的结构，您可以选择仅创建一个团队定义，并将团队数量设置为“2”。在这种情况下，如果您将团队命名为“aliens”，那么为您的团队分配的名称将为“aliens_1”和“aliens_2”。

```
{
    "name": "aliens_vs_cowboys",
    "ruleLanguageVersion": "1.0",
    "playerAttributes": [{
        "name": "skill",
        "type": "number",
        "default": 10
    }],
    "teams": [{
        "name": "cowboys",
        "maxPlayers": 8,
        "minPlayers": 4
    }, {
        "name": "aliens",
        "maxPlayers": 8,
        "minPlayers": 4
    }],
    "rules": [{
        "name": "FairTeamSkill",
        "description": "The average skill of players in each team is within 10 points from
 the average skill of all players in the match",
        "type": "distance",
        // get skill values for players in each team and average separately to produce list
 of two numbers
        "measurements": [ "avg(teams[*].players.attributes[skill])" ],
        // get skill values for players in each team, flatten into a single list, and
 average to produce an overall average
        "referenceValue": "avg(flatten(teams[*].players.attributes[skill]))",
        "maxDistance": 10 // minDistance would achieve the opposite result
    }, {
        "name": "EqualTeamSizes",
        "description": "Only launch a game when the number of players in each team matches,
 e.g. 4v4, 5v5, 6v6, 7v7, 8v8",
        "type": "comparison",
        "measurements": [ "count(teams[cowboys].players)" ],
        "referenceValue": "count(teams[aliens].players)",
        "operation": "=" // other operations: !=, <, <=, >, >=
    }],
    "expansions": [{
        "target": "rules[FairTeamSkill].maxDistance",
        "steps": [{
            "waitTimeSeconds": 5,
            "value": 50
        }, {
            "waitTimeSeconds": 15,
            "value": 100
        }]
    }]
}
```

示例 2：创建水平不对等的团队（猎人对战怪物）

此示例描述了一组玩家搜捕单个怪物的游戏模式。玩家可以选择猎人或怪物角色。猎人指定他们希望对战的怪物的最低技能水平。可随时间推移放宽猎人团队的最小规模以完成对战游戏。此方案规定了以下说明：

- 创建一个包含五个猎人的团队。
- 创建一个包含单个怪物的团队。
- 包含以下玩家属性：
  - 玩家的技能水平 (如果未提供，则默认为 10)。
  - 玩家首选的怪物技能水平 (如果未提供，则默认为 10)。
  - 玩家是否想成为怪物 (如果未提供，则默认为 0 或 false)。
- 根据以下条件选择将成为怪物的玩家：
  - 玩家必须请求怪物角色。
  - 玩家必须满足或超过已添加到猎人团队的玩家的首选最高技能水平。
- 根据以下条件选择将加入猎人团队的玩家：
  - 请求怪物角色的玩家无法加入猎人团队。
  - 如果怪物角色已填充，玩家必须要有低于建议怪物技能的怪物技能水平。
- 如果对战游戏未快速填满，则放宽猎人团队的最小规模，如下所示：
  - 30 秒后，允许启动猎人团队只包含四个玩家的游戏。
  - 60 秒后，允许启动猎人团队只包含三个玩家的游戏。

使用此规则集的说明：

- 通过为猎人和怪物分别创建两个单独的团队，您可以根据不同的条件评估团队成员。

```
{
    "name": "players_vs_monster_5_vs_1",
    "ruleLanguageVersion": "1.0",
    "playerAttributes": [{
        "name": "skill",
        "type": "number",
        "default": 10
    },{
        "name": "desiredSkillOfMonster",
        "type": "number",
        "default": 10
    },{
        "name": "wantsToBeMonster",
        "type": "number",
        "default": 0
    }],
    "teams": [{
        "name": "players",
        "maxPlayers": 5,
        "minPlayers": 5
    }, {
        "name": "monster",
        "maxPlayers": 1,
        "minPlayers": 1
    }],
    "rules": [{
        "name": "MonsterSelection",
        "description": "Only users that request playing as monster are assigned to the
 monster team",
        "type": "comparison",
```

```
        "measurements": ["teams[monster].players.attributes[wantsToBeMonster]"],
        "referenceValue": 1,
        "operation": "="
    },{
        "name": "PlayerSelection",
        "description": "Do not place people who want to be monsters in the players team",
        "type": "comparison",
        "measurements": ["teams[players].players.attributes[wantsToBeMonster]"],
        "referenceValue": 0,
        "operation": "="
    },{
        "name": "MonsterSkill",
        "description": "Monsters must meet the skill requested by all players",
        "type": "comparison",
        "measurements": ["avg(teams[monster].players.attributes[skill])"],
        "referenceValue": "max(teams[players].players.attributes[desiredSkillOfMonster])",
        "operation": ">="
    }],
    "expansions": [{
        "target": "teams[players].minPlayers",
        "steps": [{
            "waitTimeSeconds": 30,
            "value": 4
        },{
            "waitTimeSeconds": 60,
            "value": 3
        }]
    }]
}
```

示例 3：设置团队级要求和延迟限制

此示例说明如何设置玩家团队，并为每个团队应用规则集，而不是为每个玩家应用规则集。它使用单个定义创建三个平均的对战团队。它还会设置所有玩家的最大延迟。随着时间的推移可以放宽延迟最大值以完成对战游戏。此方案规定了以下说明：

- 创建三个玩家团队。
  - 每个团队包含三到五个玩家。
  - 最终团队必须包含数量相同或基本相同的玩家 (一个团队中)。
- 包含以下玩家属性：
  - 玩家的技能水平 (如果未提供，则默认为 10)。
  - 玩家的人物角色 (如果未提供，则默认为“peasant”)。
- 选择与对战游戏中其他玩家技能水平相当的玩家。
  - 确保每个团队的平均玩家技能在 10 点以内。
- 将团队数量限制为“medic”角色的以下数量：
  - 整个对战游戏最多可以包含 5 个医生。
- 仅匹配报告 50 毫秒或更少延迟的玩家。
- 如果对战游戏未快速填满，则放宽玩家的延迟要求，如下所示：
  - 10 秒之后，允许将玩家的延迟时间值增加到 100 毫秒。
  - 20 秒之后，允许将玩家的延迟时间值增加到 150 毫秒。

使用此规则集的说明：

- 规则集可确保根据玩家技能构建对战团队。为评估 `FairTeamSkill` 规则，FlexMatch 会暂时将潜在玩家加入团队并计算团队中玩家的平均技能。然后，它将规则与两个团队中的玩家平均技能进行比较。如果规则失败，则不会将潜在玩家添加到对战游戏。

- 团队和对战游戏级别的要求 (医生总数) 通过一组规则实现。此规则类型会获取所有玩家的角色属性列表，并针对最大数量进行检查。使用 `flatten` 为所有团队中的所有玩家创建列表。
- 根据延迟进行评估时，请注意以下几点：
  - 延迟数据在对战请求中作为玩家对象的一部分提供。它不是玩家属性，因此无需列出。
  - 对战构建器按区域评估延迟。延迟高于最大延迟的任何区域都将被忽略。要让对战游戏接受，玩家必须至少有一个区域的延迟低于最大延迟。
  - 如果对战请求忽略一个或多个玩家的延迟数据，则会拒绝所有对战游戏的请求。

```
{
    "name": "three_team_game",
    "ruleLanguageVersion": "1.0",
    "playerAttributes": [{
        "name": "skill",
        "type": "number",
        "default": 10
    },{
        "name": "character",
        "type": "string_list",
        "default": [ "peasant" ]
    }],
    "teams": [{
        "name": "trio",
        "minPlayers": 3,
        "maxPlayers": 5,
        "quantity": 3
    }],
    "rules": [{
        "name": "FairTeamSkill",
        "description": "The average skill of players in each team is within 10 points from
 the average skill of players in the match",
        "type": "distance",
        // get players for each team, and average separately to produce list of 3
        "measurements": [ "avg(teams[*].players.attributes[skill])" ],
        // get players for each team, flatten into a single list, and average to produce
 overall average
        "referenceValue": "avg(flatten(teams[*].players.attributes[skill]))",
        "maxDistance": 10 // minDistance would achieve the opposite result
    }, {
        "name": "CloseTeamSizes",
        "description": "Only launch a game when the team sizes are within 1 of each other.
 e.g. 3 v 3 v 4 is okay, but not 3 v 5 v 5",
        "type": "distance",
        "measurements": [ "max(count(teams[*].players))"],
        "referenceValue": "min(count(teams[*].players))",
        "maxDistance": 1
    }, {
        "name": "OverallMedicLimit",
        "description": "Don't allow more than 5 medics in the game",
        "type": "collection",
        // This is similar to above, but the flatten flattens everything into a single
        // list of characters in the game.
        "measurements": [ "flatten(teams[*].players.attributes[character])"],
        "operation": "contains",
        "referenceValue": "medic",
        "maxCount": 5
    }, {
        "name": "FastConnection",
        "description": "Prefer matches with fast player connections first",
        "type": "latency",
        "maxLatency": 50
    }],
    "expansions": [{
```

```
        "target": "rules[FastConnection].maxLatency",
        "steps": [{
            "waitTimeSeconds": 10,
            "value": 100
        }, {
            "waitTimeSeconds": 20,
            "value": 150
        }]
    }]
}
```

示例 4：使用显式排序查找最佳匹配

此示例设置了包含三个玩家的两个团队的简单对战游戏。它介绍了如何使用显式排序规则才能尽快找到最佳对战游戏。这些规则对所有活动的对战票证进行排序，以根据特定的关键要求创建最佳对战游戏。此方案根据以下说明实现：

- 创建两个玩家团队。
- 每个团队只包含三个玩家。
- 包含以下玩家属性：
  - 经验等级 (如果未提供，则默认为 50)。
  - 首选游戏模式 (可以列出多个值) (如果未提供，则默认为“coop”和“deathmatch”)。
  - 首选游戏地图，包括地图名称和首选权重 (如果未提供，则默认使用 `"defaultMap"` 和权重 100)。
- 设置预排序：
  - 根据与基准点玩家相同的游戏地图的首选项来对玩家排序。玩家可以有多个最喜欢的游戏地图，因此本示例使用首选项值。
  - 根据玩家与基准点玩家的经验等级的接近程度对玩家排序。采用这种排序方式，所有玩家在所有团队中的经验等级会尽可能地接近。
- 所有团队中的所有玩家必须至少选择一个共同的游戏模式。
- 所有团队中的所有玩家必须至少选择一个共同的游戏地图。

使用此规则集的说明：

- 游戏地图排序采用与 mapPreference 属性值进行比较的绝对排序。由于它是规则集中的第一个，此排序将首先执行。
- 经验排序采用比较潜在玩家的技能级别与基准点玩家的技能的距离排序。
- 排序按它们在规则集中的排列顺序执行。在这种情况下，玩家先按游戏地图首选项进行排序，然后按经验等级排序。

```
{
    "name": "multi_map_game",
    "ruleLanguageVersion": "1.0",
    "playerAttributes": [{
        "name": "experience",
        "type": "number",
        "default": 50
    }, {
        "name": "gameMode",
        "type": "string_list",
        "default": [ "deathmatch", "coop" ]
    }, {
        "name": "mapPreference",
        "type": "string_number_map",
        "default": { "defaultMap": 100 }
    }, {
```

```
        "name": "acceptableMaps",
        "type": "string_list",
        "default": [ "defaultMap" ]
    }],
    "teams": [{
        "name": "red",
        "maxPlayers": 3,
        "minPlayers": 3
    }, {
        "name": "blue",
        "maxPlayers": 3,
        "minPlayers": 3
    }],
    "rules": [{
        // We placed this rule first since we want to prioritize players preferring the
 same map
        "name": "MapPreference",
        "description": "Favor grouping players that have the highest map preference aligned
 with the anchor's favorite",
        // This rule is just for sorting potential matches.  We sort by the absolute value
 of a field.
        "type": "absoluteSort",
        // Highest values go first
        "sortDirection": "descending",
        // Sort is based on the mapPreference attribute.
        "sortAttribute": "mapPreference",
        // We find the key in the anchor's mapPreference attribute that has the highest
 value.
        // That's the key that we use for all players when sorting.
        "mapKey": "maxValue"
    }, {
        // This rule is second because any tie-breakers should be ordered by similar
 experience values
        "name": "ExperienceAffinity",
        "description": "Favor players with similar experience",
        // This rule is just for sorting potential matches.  We sort by the distance from
 the anchor.
        "type": "distanceSort",
        // Lowest distance goes first
        "sortDirection": "ascending",
        "sortAttribute": "experience"
    }, {
        "name": "SharedMode",
        "description": "The players must have at least one game mode in common",
        "type": "collection",
        "operation": "intersection",
        "measurements": [ "flatten(teams[*].players.attributes[gameMode])"],
        "minCount": 1
    }, {
        "name": "MapOverlap",
        "description": "The players must have at least one map in common",
        "type": "collection",
        "operation": "intersection",
        "measurements": [ "flatten(teams[*].players.attributes[acceptableMaps])"],
        "minCount": 1
    }]
}
```

示例 5：查找多个玩家属性的交集

此示例说明如何使用集合规则，在两个或更多玩家属性中查找交集。在处理集合时，您可以对单个属性使用 `intersection` 操作，并对多个属性使用 `reference_intersection_count` 操作。

为了说明这种方法，此示例会在对战游戏中根据玩家的角色首选项来评估玩家。该示例游戏是一个“混战”风格的游戏，对战游戏中的所有玩家都是对手。每个玩家需要 (1) 为自己选择一个角色，(2) 选择他们希望作为对手的角色。我们需要一个规则，该规则可确保对战游戏中的每位玩家所使用的角色都位于所有其他玩家的首选对手列表中。

该示例规则集描述了一个具有以下特征的对战游戏：

- 团队结构：一个团队有 5 个玩家
- 玩家属性：
  - myCharacter：玩家的选定角色。
  - preferredOpponents：玩家希望作为对手的角色的列表。
- 匹配规则：如果所使用的每个角色都位于每个玩家的首选对手列表中，则潜在的对战是可以接受的。

为了实施该对战游戏规则，此示例使用具有以下属性值的集合规则：

- 操作 – 使用 `reference_intersection_count` 操作来评估测量值中的每个字符串列表与参考值中的字符串列表的相交情况。
- 测量值 – 使用 `flatten` 属性表达式创建一个字符串列表列表，其中每个字符串列表包含一个玩家的 myCharacter 属性值。
- 参考值 – 使用 `set_intersection` 属性表达式创建对战游戏中每个玩家通用的所有 preferredOpponents 属性值的字符串列表。
- 限制 – `minCount` 设置为 1，以确保每个玩家的选定角色（测量值中的一个字符串列表）匹配至少一个对所有玩家通用的首选对手。（参考值中的字符串）。
- 扩展 – 如果对战游戏在 15 秒内未满员，则放松最小交集要求。

此规则的流程如下：

1. 将一位玩家添加到潜在对战游戏。重新计算参考值 (字符串列表)，以便包含与新玩家的首选对手列表的交集。重新计算测量值 (字符串列表的列表)，以便将新玩家的选定角色作为新的字符串列表添加。
2. Amazon GameLift 验证测量值 (玩家的选定角色) 中的每个字符串列表与参考值 (玩家的首选对手) 中的至少一个字符串相交。在本示例中，由于测量值中的每个字符串列表中仅包含一个值，因此交集为 0 或 1。
3. 如果测量值中的任何字符串列表与参考值字符串列表不相交，则该规则失败，并且从潜在对战游戏中删除该新玩家。
4. 如果对战游戏在 15 秒内未满员，则放弃对手匹配要求，以便在对战游戏中填满剩余的玩家位置。

```
{
    "name": "preferred_characters",
    "ruleLanguageVersion": "1.0",

    "playerAttributes": [{
        "name": "myCharacter",
        "type": "string_list"
    }, {
        "name": "preferredOpponents",
        "type": "string_list"
    }],

    "teams": [{
        "name": "red",
        "minPlayers": 5,
        "maxPlayers": 5
    }],

    "rules": [{
```

```
        "description": "Make sure that all players in the match are using a character that
 is on all other players' preferred opponents list.",
        "name": "OpponentMatch",
        "type": "collection",
        "operation": "reference_intersection_count",
        "measurements": ["flatten(teams[*].players.attributes[myCharacter])"],
        "referenceValue":
 "set_intersection(flatten(teams[*].players.attributes[preferredOpponents]))",
        "minCount":1
    }],
    "expansions": [{
        "target": "rules[OpponentMatch].minCount",
        "steps": [{
            "waitTimeSeconds": 15,
            "value": 0
        }]
    }]
}
```

示例 6：比较所有玩家的属性

此示例说明如何在一组玩家之间比较玩家属性。

该示例规则集描述了一个具有以下特征的对战游戏：

- 团队结构：两个单人游戏团队
- 玩家属性：
  - gameMode：玩家选择的游戏类型（如果未提供，则默认为“基于旋转”）。
  - gameMap：玩家选择的游戏世界（如果未提供，则默认为 1）。
  - 字符：玩家选择的角色（无默认值，表示玩家必须指定角色）。
- 匹配规则：匹配的玩家必须满足以下要求：
  - 玩家必须选择相同的游戏模式。
  - 玩家必须选择相同的游戏地图。
  - 玩家必须选择不同的角色。

使用此规则集的说明：

- 为了实施该对战游戏规则，此示例使用比较规则来检查所有玩家的属性值。对于游戏模式和地图，该规则验证值是相同的。对于角色，该规则验证值是不同的。
- 此示例使用一个玩家定义与数量属性来创建两个玩家团队。为团队分配了下列名称：“player_1”和“player_2”。

```
{
    "name": "",
    "ruleLanguageVersion": "1.0",

    "playerAttributes": [{
        "name": "gameMode",
        "type": "string",
        "default": "turn-based"
    }, {
        "name": "gameMap",
        "type": "number",
        "default": 1
    }, {
        "name": "character",
        "type": "number"
```

```
    }],

    "teams": [{
        "name": "player",
        "minPlayers": 1,
        "maxPlayers": 1,
        "quantity": 2
    }],

    "rules": [{
        "name": "SameGameMode",
        "description": "Only match players when they choose the same game type",
        "type": "comparison",
        "operation": "=",
        "measurements": ["flatten(teams[*].players.attributes[gameMode])"]
    }, {
        "name": "SameGameMap",
        "description": "Only match players when they're in the same map",
        "type": "comparison",
        "operation": "=",
        "measurements": ["flatten(teams[*].players.attributes[gameMap])"]
    }, {
        "name": "DifferentCharacter",
        "description": "Only match players when they're using different characters",
        "type": "comparison",
        "operation": "!=",
        "measurements": ["flatten(teams[*].players.attributes[character])"]
    }]
}
```

示例 7：创建大型对战

此示例说明如何为超过 40 位玩家的对战设置规则集。当规则集描述 maxPlayer 总数超过 40 的团队时，该团队会作为大型对战进行处理。在 设计 FlexMatch 大型对战规则集 (p. 18) 中了解更多信息。

此示例规则集使用以下说明创建对战：

- 创建一个最多有 200 位玩家的团队，玩家人数最低要求为 175 位。
- 平衡条件：根据类似的技能级别选择玩家。所有玩家均必须报告其技能级别才能被匹配。
- 批处理首选项：在创建对战游戏时，按类似的平衡条件分组玩家。
- 延迟规则：将可接受的最大玩家延迟设置为 150 毫秒。
- 如果未能快速填充对战游戏，则放宽要求以在合理的时间内完成匹配。
  - 10 秒后，接受有 150 位玩家的团队。
  - 12 秒后，将可接受的最大延迟提高到 200 毫秒。
  - 15 秒后，接受有 100 位玩家的团队。

使用此规则集的说明：

- 由于算法使用“最大群体”批处理首选项，玩家会根据平衡条件排序。因此，对战往往填得更满，包含技能更加类似的玩家。所有玩家均符合可接受的延迟要求，但可能无法在他们的位置获得可能的最佳延迟。
- 此规则集中使用的算法策略“最大群体”是默认设置。要使用默认设置，您可以选择忽略此设置。
- 如果您已启用了对战回填，请不要太快地放宽玩家计数要求，否则，您最后可能会有太多部分填充的游戏会话。在 放宽大型对战要求 (p. 20) 中了解更多信息。

```
{
    "name": "free-for-all",
    "ruleLanguageVersion": "1.0",
```

```
    "playerAttributes": [{
        "name": "skill",
        "type": "number"
    }],
    "algorithm": {
        "balancedAttribute": "skill",
        "strategy": "balanced",
        "batchingPreference": "largestPopulation"
    },
    "teams": [{
        "name": "Marauders",
        "maxPlayers": 200,
        "minPlayers": 175
    }],
    "rules": [{
        "name": "low-latency",
        "description": "Sets maximum acceptable latency",
        "type": "latency",
        "maxLatency": 150
    }],
    "expansions": [{
        "target": "rules[low-latency].maxLatency",
        "steps": [{
            "waitTimeSeconds": 12,
            "value": 200
        }],
        "target": "teams[Marauders].minPlayers",
        "steps": [{
            "waitTimeSeconds": 10,
            "value": 150
        }, {
            "waitTimeSeconds": 15,
            "value": 100
        }]
    }]
}
```

示例 8：创建多团队大型对战

此示例说明如何为有多个团队可能超过 40 位玩家的对战设置规则集。此示例说明如何使用一个定义创建多个相同的团队，以及规模不对称的团队如何在对战创建过程中填充。

此示例规则集使用以下说明创建对战：

- 创建十个相同的“猎人”团队，每个团队最多有 15 位玩家，并创建一个刚好有 5 位玩家的“怪物”团队。
- 平衡条件：根据怪物技能的数量选择玩家。如果玩家未报告其技能计数，请使用默认值 5。
- 批处理首选项：根据玩家报告最快玩家延迟的区域分组玩家。
- 延迟规则：将可接受的最大玩家延迟设置为 200 毫秒。
- 如果未能快速填充对战游戏，则放宽要求以在合理的时间内完成匹配。
  - 15 秒后，接受有 10 位玩家的团队。
  - 20 秒后，接受有 8 位玩家的团队。

使用此规则集的说明：

- 此规则集定义了可能容纳最多 155 位玩家的团队，这使其成为大型对战。（10 x 15 位猎人 + 5 个怪物 = 155）
- 由于算法使用“最快区域”批处理首选项，玩家更可能被置于他们报告更快延迟的区域，而不是他们报告较高（但可接受）延迟的区域。同时，可能有更少玩家的对战和平衡条件（怪物技能的数量）可能会变化更大。

- 当为多团队定义（数量 > 1）定义了扩展时，扩展将应用于使用该定义创建的所有团队。因此，通过放宽猎人团队的最低玩家设置，全部十个猎人团队都会受到均等的影响。
- 由于此规则集经过优化以最大程度地减少玩家延迟，延迟规则将充当“捕获全部”方法来排除没有可接受连接选项的玩家。我们不需要放宽此要求。
- 下面介绍了 FlexMatch 在有任何扩展生效之前如何针对此规则集填充对战：
  - 还没有任何团队达到 minPlayers 计数。猎人团队有 15 个空闲位置，怪物团队有 5 个空闲位置。
    - 前 100 个玩家被分配（每个团队 10 个）到十个猎人团队。
    - 接下来 22 个玩家按顺序分配（每个团队 2 个）到猎人团队和怪物团队。
  - 猎人团队已达到 minPlayers 计数，每个团队有 12 个玩家。怪物团队有 2 个玩家，还未达到 minPlayers 计数。
    - 接下来的三个玩家被分配到怪物团队。
  - 所有团队均已达到 minPlayers 计数。每个猎人团队都有三个空闲位置。怪物团队已满。
    - 最后 30 个玩家按顺序分配到猎人团队，确保所有猎人团队具有几乎相同的规模（加上或减去一个玩家）。
- 如果您已为使用此规则集创建的对战启用了回填，请不要太快地放宽玩家计数要求，否则，您最后可能会有太多部分填充的游戏会话。在 放宽大型对战要求 (p. 20) 中了解更多信息。

```
{
    "name": "monster-hunters",
    "ruleLanguageVersion": "1.0",
    "playerAttributes": [{
        "name": "monster-kills",
        "type": "number",
        "default": 5
    }],
    "algorithm": {
        "balancedAttribute": "monster-kills",
        "strategy": "balanced",
        "batchingPreference": "fastestRegion"
    },
    "teams": [{
        "name": "Monsters",
        "maxPlayers": 5,
        "minPlayers": 5
    }, {
        "name": "Hunters",
        "maxPlayers": 15,
        "minPlayers": 12,
        "quantity": 10
    }],
    "rules": [{
        "name": "latency-catchall",
        "description": "Sets maximum acceptable latency",
        "type": "latency",
        "maxLatency": 150
    }],
    "expansions": [{
        "target": "teams[Hunters].minPlayers",
        "steps": [{
            "waitTimeSeconds": 15,
            "value": 10
        }, {
            "waitTimeSeconds": 20,
            "value": 8
        }]
    }]
}
```

设置 FlexMatch 事件通知

如果您在游戏中使用 FlexMatch 对战，您需要一种方法来跟踪单独对战请求的状态并相应采取操作。有时候，例如当玩家需要接受建议的对战游戏时，这些操作非常注重时间。实施事件通知是跟踪对战事件的快速有效的方法。所有投入实际生产的游戏，或具有大量对战活动的预生产中的游戏都应使用事件通知。

有两个选项可用于设置事件通知。您可以使用 Amazon CloudWatch Events，其中包括一套工具，可用于管理事件和对其采取行动。此外，您可以设置自己的 SNS 主题并将其附加到您的对战构建器，以用于直接接收对战事件通知。

请参阅 FlexMatch 对战事件 (p. 58)中由 Amazon GameLift 发出的 FlexMatch 事件的列表。

设置 CloudWatch Events

Amazon GameLift 自动将所有对战事件发送到 CloudWatch Events。使用 CloudWatch Events，您可以设置规则，让对战事件路由到一系列目标以进行处理，包括 SNS 主题和其他 AWS 服务。例如，您可以设置一个规则，将事件“PotentialMatchCreated”路由到处理玩家接受情况的 AWS Lambda 函数。在包括教程集合的 CloudWatch Events入门指南中详细了解如何使用 。

如果您在配置您的对战构建器时计划使用 CloudWatch Events，可以将通知目标字段留空；如果您希望使用两个选项，则参考 SNS 主题。

要在 CloudWatch Events 中访问 Amazon GameLift 对战事件，请转到 Amazon CloudWatch 控制台并打开 Events。请确保您所在的区域是设置自己的对战配置的区域。有关获取账户凭证以访问 CloudWatch Events 的更多信息，请参阅登录 Amazon CloudWatch 控制台。每个对战事件由服务 (GameLift)、对战名称和对战票证标识。

设置 SNS 主题

您可以要求 Amazon GameLift 将 FlexMatch 对战构建器生成的所有事件发布到 Amazon Simple Notification Service (SNS) 主题。配置对战构建器时，将通知目标字段设置为 SNS 主题 ARN。

为 Amazon GameLift 事件通知设置 SNS 主题

1. 转到 Amazon Simple Notification Service 控制台。
2. 创建主题。 在 SNS 控制面板中，选择 Create topic (创建主题)，然后按照说明创建主题。创建主题之后，控制台自动打开新主题的 Topic details 页面。
3. 允许 Amazon GameLift 向主题发布消息。 如果您尚未打开主题的“Topic Details”页面，请在导航栏中选择“Topics”，然后单击主题 ARN 以打开它。选择主题操作 Edit topic policy，然后转到 Advanced view tab。

   将以下加粗的语法添加到现有策略的结尾。(为清晰起见显示了整个策略。)

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
        "SNS:AddPermission",
```

```
        "SNS:RemovePermission",
        "SNS:DeleteTopic",
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:Publish",
        "SNS:Receive"
      ],
      "Resource": "arn:aws-cn:sns:your_region:your_account:your_topic_name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceOwner": "your_account"
        }
      }
    },
    {
      "Sid": "__console_pub_0",
      "Effect": "Allow",
      "Principal": {
        "Service": "gamelift.amazonaws.com"
      },
      "Action": "SNS:Publish",
      "Resource": "arn:aws-cn:sns:your_region:your_account:your_topic_name"
    }
  ]
}
```

为 FlexMatch 准备游戏

使用 GameLift FlexMatch 向游戏添加玩家对战功能。FlexMatch 可用于适用于自定义游戏服务器和 实时服务器 的托管 GameLift 解决方案。

FlexMatch 可将对战服务与自定义规则引擎搭配使用。这样，您就可以设计如何根据适用于您的游戏的玩家属性和游戏模式匹配对战玩家，然后依靠 FlexMatch 来管理构成玩家组的基本要素，并将它们放置到游戏中。请参阅FlexMatch 规则集示例 (p. 22)中有关自定义对战的更多详细信息。

FlexMatch 基于队列功能进行构建。对战游戏组成之后，FlexMatch 将对战游戏的详细信息处理成供您选择的队列。此队列可在您的 Amazon GameLift 队组中搜索可用的托管资源，并为对战游戏启动新的游戏会话。

此部分的主题介绍如何向您的游戏服务器和游戏客户端添加对战支持。要为游戏创建对战构建器，请参阅 构建 GameLift FlexMatch 对战构建器 (p. 9)。有关 FlexMatch 如何运行的更多信息，请参阅 Amazon GameLift FlexMatch 的工作原理 (p. 2)。

将 FlexMatch 添加到游戏客户端

本主题介绍如何为您的客户端游戏服务添加 FlexMatch 对战支持。此过程与将 FlexMatch 与 GameLift 托管托管托管托管托管托管或与其他托管解决方案结合使用时基本相同。要了解有关 FlexMatch 以及如何为游戏设置自定义对战构建器的更多信息，请参阅以下主题：

- FlexMatch 与 GameLift 托管的集成 (p. 8)
- Amazon GameLift FlexMatch 的工作原理 (p. 2)
- 构建 GameLift FlexMatch 对战构建器 (p. 9)
- FlexMatch 规则集示例 (p. 22)

要在游戏中启用 FlexMatch 对战，请添加以下功能：

- 准备为一个或多个玩家请求对战（必需）。
- 跟踪对战请求的状态（必需）。
- 要求玩家接受建议的对战游戏（可选）。
- 为新对战游戏创建游戏会话之后，获取玩家连接信息并加入游戏。

准备为玩家请求对战

我们强烈建议您的游戏客户端通过客户端游戏服务发出对战请求。通过使用可信源，您可以更轻松地防范黑客攻击和虚假玩家数据。如果您的游戏具有会话目录服务，那么这是一个用于处理对战请求的好选项。

要准备您的客户端服务，请执行以下任务：

- 添加 GameLift API。 您的客户端服务使用 GameLift API 中的功能，它是 AWS SDK 的一部分。请参阅适用于客户端服务的 GameLift SDKs以了解有关 AWS 开发工具包的更多信息并下载最新版本。将此 SDK 添加到您的客户端服务项目。
- 设置对战票证系统。 必须向所有对战请求分配一个唯一票证 ID。您需要一个生成唯一 IDs 并将其分配给新对战请求的机制。票证 ID 可以使用任意字符串格式，最多 128 个字符。
- 获取对战构建器信息。 获取您计划要使用的对战配置的名称。您还需要对战构建器的必需玩家属性列表，这在对战构建器规则集中定义。
- 获取玩家数据。 设置获取各个玩家相关数据的方法。这包括玩家 ID、玩家属性值，以及玩家可能接入游戏的各个区域的更新延迟数据。

- （可选）启用对战回填。 确定您想要如何回填现有的匹配游戏。如果您的对战构建器将回填模式设置为“手动”，您可能需要为您的游戏添加回填支持。如果回填模式设置为“自动”，您可能需要一种为单个游戏会话关闭此模式的方法。在 使用 FlexMatch 回填现有游戏 (p. 41) 中了解有关管理对战回填的更多信息。

请求玩家对战

将代码添加到您的客户端服务来创建和管理 FlexMatch 对战构建器的对战请求。对于将 FlexMatch 与 FlexMatch 托管托管托管托管的游戏以及将 GameLift 用作独立解决方案的游戏，请求 FlexMatch 对战的过程是相同的。

创建对战请求：

- 调用 GameLift API StartMatchmaking。 每个请求都必须包含以下信息。

  对战构建器

  要用于请求的对战配置的名称。FlexMatch 将各个请求放置到指定对战构建器的池中，并将根据对战构建器的配置方式来处理请求。这包括强制施加时间限制，是否请求玩家接受匹配，在放置生成的游戏会话时使用哪个队列，等等。在 设计 FlexMatch 对战构建器 (p. 9) 中了解有关对战构建器和规则集的更多信息。

  票证 ID

  分配给请求的唯一票证 ID。与请求相关的所有信息（包括事件和通知）都将引用票证 ID。

  玩家数据

  您要为其创建对战的玩家的列表。如果根据对战规则和延迟最低值，请求中的任意玩家不满足对战要求，则对战请求绝不会生成成功的对战。您可在一个对战请求中包括最多十位玩家。当一个请求中有多个玩家时，FlexMatch 尝试创建单个对战并将所有玩家分配到相同团队中（随机选择）。如果请求包含的玩家数太多，无法放在一个对战团队中，则对战请求失败。例如，如果您设置了对战构建器，以创建 2 对 2 对战（两个团队，每个团队两个玩家），您无法发送包含两个以上玩家的对战请求。

  > **Note**
  >
  > 一个玩家 (通过其玩家 ID 标识) 一次只能包括在一个有效对战请求中。在您为玩家创建新请求时，将自动取消任何具有相同玩家 ID 的有效对战票证。

  对于每个列出的玩家，请提供以下数据：

  - 玩家 ID – 每个玩家必须具有一个唯一的玩家 ID，该 ID 由您生成。请参阅生成玩家IDs。
  - 玩家属性 – 如果所使用的对战构建器需要玩家属性，请求必须为每个玩家提供这些属性。必需的玩家属性在对战构建器的规则集中定义，同时还会指定属性的数据类型。玩家属性仅在规则集指定属性的默认值时是可选的。如果对战请求未提供所有玩家的必需玩家属性，对战请求可能永远无法成功。在构建 FlexMatch 规则集 (p. 13)和FlexMatch 规则集示例 (p. 22)中了解有关对战构建器规则集和玩家属性的更多信息。
  - 玩家延迟 – 如果所使用的对战构建器有玩家延迟规则，请求必须报告每个玩家的延迟。玩家延迟数据是显示每个玩家的一个或多个值的列表。它表示对战构建器的队列中各个区域的玩家体验的延迟。如果请求中未包含延迟值，玩家将无法匹配，请求将失败。

检索对战请求详细信息：

- 发送对战请求后，您可以通过调用包含请求票证 ID 的 DescribeMatchmaking 来查看请求详细信息。此调用将返回请求信息，包括当前状态。成功完成请求之后，票证还将包含游戏客户端连接到对战所需的信息。

取消对战请求：

- 您可以随时通过调用包含请求票证 ID 的 StopMatchmaking 取消对战请求。

跟踪对战事件

将代码添加到客户端服务以跟踪与对战请求相关的所有事件。对战事件包括

> Note
>
> 在使用大量对战场景之前，您应该使用事件通知设置游戏，例如进行预生产负载测试。公开发布版中的所有游戏都应该使用通知，而不考虑容量。连续轮询方法仅适用于对战使用率较低的开发中的游戏。

设置通知，以跟踪 GameLift 为对战过程发出的事件。您可以直接设置通知，也可以通过创建 SNS 主题或使用 Amazon EventBridge 来设置通知。有关设置通知的更多信息，请参阅设置 FlexMatch 事件通知 (p. 34)。设置通知之后，请在客户端服务上添加侦听器以检测事件并根据需要做出响应。

在经过相当长一段时间而未通知的情况下，最好定期轮询状态更新来作为通知的备用手段。为了最大限度地减少对对战性能的影响，请务必在提交对战票证后或最后一次收到通知后，等待至少 30 秒后再轮询。

通过调用包含请求票证 ID 的 DescribeMatchmaking 来检索对战请求票证，包括当前状态。我们建议轮询频率不要超过每 10 秒一次。此方法仅在低容量开发场景中使用。

请求玩家接受

如果您使用的是开启了玩家接受的对战构建器，请将代码添加到您的客户端服务来管理玩家接受过程。对于那些将 FlexMatch 与 GameLift 托管托管托管托管托管的托管结合使用的游戏以及将 FlexMatch 用作独立解决方案的游戏，管理玩家接受的过程是相同的。

要求玩家接受建议的对战游戏：

1. 检测建议的对战游戏何时需要玩家接受。 监控对战票证以检测状态更改为 `REQUIRES_ACCEPTANCE` 的时间。 对此状态的更改将触发 FlexMatch 事件 `MatchmakingRequiresAcceptance`。
2. 从所有玩家获取接受信息。 创建一个机制，以在对战票证中向每个玩家呈现建议的对战游戏详细信息。玩家必须能够表明他们接受或拒绝建议的对战游戏。您可以通过调用 DescribeMatchmaking 来检索对战游戏详细信息。 在对战构建器撤消建议的对战游戏之前，玩家的响应时间有限。
3. 向 FlexMatch 报告玩家响应。 通过调用接受或拒绝的 AcceptMatch 来报告玩家响应。对战请求中的所有玩家必须接受对战游戏才能继续。
4. 处理具有失败接受的票证。 当建议的对战游戏中的任何一个玩家拒绝对战游戏，或者未能在接受时限内响应时，请求失败。已接受对战的玩家的票证将自动返回到票证池中。未接受对战游戏的玩家的票证将转为 FAILURE 状态，并且不再进行处理。对于有多个玩家的票证，如果票证中的任何玩家未接受对战游戏，整个票证将失败。

连接到对战游戏

将代码添加到您的客户端服务以处理格式成功的对战（状态 `COMPLETED` 或事件 `MatchmakingSucceeded`）。这包括向游戏客户端通知对战游戏的玩家和传递连接信息。

对于使用 GameLift 托管托管的游戏，在成功完成对战请求时，游戏会话连接信息将添加到对战票证。通过调用 DescribeMatchmaking 检索已完成的对战票证。 连接信息包括游戏会话的 IP 地址和端口，以及每个玩家 ID 的玩家会话 ID。在GameSessionConnectionInfo中了解更多信息。 您的游戏客户端可以使用此信息直接

连接到对战的游戏会话。连接请求应包含玩家会话 ID 和玩家 ID。此数据将连接的玩家与游戏会话的对战数据相关联，其中包括团队分配 （请参阅 GameSession ）。

对于使用其他托管解决方案（包括 GameLift FleetIQ ）的游戏，您必须构建一个机制以允许对战玩家连接到适当的游戏会话。

示例对战请求

以下代码段为多个不同的对战构建器生成对战请求。如文中所述，请求必须提供所使用的对战构建器需要的玩家属性（在对战构建器的规则集中定义）。提供的属性必须使用在规则集中定义的相同的数据类型：数字 (N) 或字符串 (S)。

```
# Uses matchmaker for two-team game mode based on player skill level
def start_matchmaking_for_cowboys_vs_aliens(config_name, ticket_id, player_id, skill,
 team):
    response = gamelift.start_matchmaking(
        ConfigurationName=config_name,
        Players=[{
            "PlayerAttributes": {
                "skill": {"N": skill}
            },
            "PlayerId": player_id,
            "Team": team
        }],
        TicketId=ticket_id)

# Uses matchmaker for monster hunter game mode based on player skill level
def start_matchmaking_for_players_vs_monster(config_name, ticket_id, player_id, skill,
 is_monster):
    response = gamelift.start_matchmaking(
        ConfigurationName=config_name,
        Players=[{
            "PlayerAttributes": {
                "skill": {"N": skill},
                "desiredSkillOfMonster": {"N": skill},
                "wantsToBeMonster": {"N": int(is_monster)}
            },
            "PlayerId": player_id
        }],
        TicketId=ticket_id)

# Uses matchmaker for brawler game mode with latency
def start_matchmaking_for_three_team_brawler(config_name, ticket_id, player_id, skill,
 role):
    response = gamelift.start_matchmaking(
        ConfigurationName=config_name,
        Players=[{
            "PlayerAttributes": {
                "skill": {"N": skill},
                "character": {"S": [role]},
            },
            "PlayerId": player_id,
            "LatencyInMs": { "us-west-2": 20}
        }],
        TicketId=ticket_id)

# Uses matchmaker for multiple game modes and maps based on player experience
def start_matchmaking_for_multi_map(config_name, ticket_id, player_id, skill, maps, modes):
    response = gamelift.start_matchmaking(
        ConfigurationName=config_name,
        Players=[{
            "PlayerAttributes": {
                "experience": {"N": skill},
                "gameMode": {"SL": modes},
```

```
            "mapPreference": {"SL": maps}
        },
        "PlayerId": player_id
    }],
    TicketId=ticket_id)
```

将 FlexMatch 添加到 GameLift 托管的游戏服务器

本主题介绍如何为使用 FlexMatch 托管托管的自定义游戏服务器添加 GameLift 对战支持。要了解有关如何向游戏添加 FlexMatch 的更多信息，请参阅以下主题：

- Amazon GameLift FlexMatch 的工作原理 (p. 2)
- FlexMatch 与 GameLift 托管的集成 (p. 8)

本主题中的信息假定您已将 GameLift 服务器开发工具包成功集成到游戏服务器项目中，如将 GameLift 添加到游戏服务器中所述。完成此工作后，您便有了所需的大部分机制。本主题中的各个部分介绍了处理使用 FlexMatch 设置的游戏需要执行的其他工作。

设置您的游戏服务器以进行对战

要设置您的游戏服务器来处理已对战的游戏，请完成以下任务。

1. 启动使用对战创建的游戏会话。 要请求新的游戏会话，GameLift 会向您的游戏服务器发送一个 `onStartGameSession()` 请求，其中包含一个游戏会话对象（请参阅 GameSession）。您的游戏服务器使用游戏会话信息 (包括自定义的游戏数据) 以启动请求的游戏会话。有关更多详细信息，请参阅启动游戏会话。

   对于已匹配的游戏，游戏会话对象还包含一组对战构建器数据。对战构建器数据包含您的游戏服务器启动新的对战游戏会话所需的信息。这包括对战的团队结构、团队分配以及可能与您的游戏相关的某些玩家属性。例如，您的游戏可以根据平均玩家技能级别解锁某些功能，或根据玩家的首选项选择地图。在使用对战构建器数据 (p. 40) 中了解更多信息。
2. 处理玩家连接。 在连接到已对战的游戏时，游戏客户端将引用玩家 ID 和玩家会话 ID（请参阅验证新玩家）。您的游戏服务器使用玩家 ID 将传入玩家与对战构建器数据中的玩家信息进行关联。对战构建器数据将标识玩家的团队分配，并且可能提供在游戏中正确地代表玩家的其他信息。
3. 在玩家离开游戏时进行报告。 确保您的游戏服务器正在调用服务器 API `RemovePlayerSession()` 来报告已退出的玩家（请参阅报告玩家会话结束）。如果您使用 FlexMatch 回填填充现有游戏中的空位置，此步骤非常重要。如果您的游戏通过客户端游戏服务发起回填请求，此步骤非常重要。在 使用 FlexMatch 回填现有游戏 (p. 41) 中了解有关实施 FlexMatch 回填的更多信息。
4. 为现有的已对战游戏会话请求新玩家（可选）。 确定您想要如何回填现有的匹配游戏。如果您的对战构建器将回填模式设置为“手动”，您可能需要为您的游戏添加回填支持。如果回填模式设置为“自动”，您可能需要一种为单个游戏会话关闭此模式的方法。例如，您可能想要在到达游戏中的某个点后停止回填游戏会话。在 使用 FlexMatch 回填现有游戏 (p. 41) 中了解有关管理对战回填的更多信息。

使用对战构建器数据

您的游戏服务器必须能够识别和使用 GameSession 对象中的游戏信息。每当启动或更新游戏会话时，GameLift 服务就会将这些对象传递到您的游戏服务器。核心游戏会话信息包括游戏会话 ID 和名称、最大玩家数量、连接信息和自定义游戏数据 (如果提供)。

对于使用 FlexMatch 创建的游戏会话，`GameSession` 对象还包含一组对战构建器数据。除了唯一的对战 ID，它将标识创建对战的对战构建器并描述团队、团队分配和玩家。它包括原始对战请求中的玩家属性 (请参阅 Player 对象)。它不包括玩家延迟；如果您需要当前玩家的延迟数据 (如对战回填)，建议您获取最新数据。

Note

对战构建器数据指定完整的对战配置 ARN，此 ARN 将标识配置名称、AWS 账户和区域。在从游戏客户端或服务请求对战回填时，仅需要配置名称。您可以通过解析出":matchmakingconfiguration/"后面的字符串来提取配置名称。在显示的示例中，对战配置名称为"MyMatchmakerConfig"。

以下 JSON 显示一组典型的对战构建器数据。此示例描述了一个两人游戏，该游戏根据技能评级和获得的最高级别匹配玩家。对战构建器还根据角色进行对战，并且确保对战的玩家至少具有一个共同的地图首选项。在这种情况下，游戏服务器应该能够确定哪个地图最受偏爱，并在游戏会话中使用它。

```
{
 "matchId":"1111aaaa-22bb-33cc-44dd-5555eeee66ff",
 "matchmakingConfigurationArn":"arn:aws-cn:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig",
 "teams":[
    {"name":"attacker",
  "players":[
           {"playerId":"4444dddd-55ee-66ff-77aa-8888bbbb99cc",
   "attributes":{
    "skills":{
     "attributeType":"STRING_DOUBLE_MAP",
     "valueAttribute":{"Body":10.0,"Mind":12.0,"Heart":15.0,"Soul":33.0}}
   }
  }]
 },{
  "name":"defender",
  "players":[{
   "playerId":"3333cccc-44dd-55ee-66ff-7777aaaa88bb",
   "attributes":{
    "skills":{
     "attributeType":"STRING_DOUBLE_MAP",
     "valueAttribute":{"Body":11.0,"Mind":12.0,"Heart":11.0,"Soul":40.0}}
   }
  }]
 }]
}
```

使用 FlexMatch 回填现有游戏

匹配回填使用 FlexMatch 机制为现有的已匹配游戏会话寻找新玩家。尽管您始终可以将玩家添加到任何游戏中（请参阅将玩家加入游戏会话），但是对战回填可确保新玩家符合与当前玩家相同的匹配条件。此外，对战回填会将新玩家分配到团队，管理玩家接受，并将更新的对战信息发送到游戏服务器。在FlexMatch 对战过程 (p. 5)中了解有关对战回填的更多信息。

Note

FlexMatch 回填当前不能使用 实时服务器 用于游戏。

回填机制有两种类型：

- 要填充以少于允许的最大玩家数开始的游戏会话，请启用自动回填。
- 要取代正在进行的游戏会话中退出的玩家，请向游戏服务器添加功能以发送回填请求。

打开自动回填

使用自动匹配回填时，每当游戏会话开始时有一个或多个玩家槽位未满，GameLift 都将自动触发回填请求。此功能允许游戏在找到最少匹配玩家数量后立即开始，并在匹配到其他玩家后填充剩余槽位。您可以随时选择停止自动回填。

例如，假设一个游戏可容纳六到十名玩家。FlexMatch 最初查找六名玩家，组成对战游戏，然后开始新的游戏会话。使用自动回填时，新游戏会话可以立即要求增加四名玩家。根据游戏风格，我们可能希望允许新玩家在游戏会话期间随时加入。或者，我们可能希望在初始设置阶段之后、游戏开始之前停止自动回填。

要向您的游戏添加自动回填，请对您的游戏进行以下更新。

1. 启用自动回填。 自动回填在对战配置中管理。启用后，它将用于使用该对战构建器创建的所有匹配的游戏会话。当游戏会话在游戏服务器上启动时，GameLift 便会开始为未满游戏会话生成回填请求。

   要打开自动回填，请打开对战配置并将回填模式设置为“AUTOMATIC”(自动)。有关更多详细信息，请参阅 创建对战配置 (p. 11)
2. 启用回填优先级。自定义您的对战过程，优先于在创建新对战游戏之前填充回填请求。在对战规则集中，添加算法组件并将回填优先级设置为“高”。有关更多信息，请参阅自定义匹配算法 (p. 14)。
3. 使用新的对战构建器数据更新游戏会话。 Amazon GameLift 使用服务器开发工具包回调函数 `onUpdateGameSession` 通过对战信息更新您的游戏服务器（请参阅初始化服务器进程）。将代码添加到游戏服务器，在回填活动后处理更新的游戏会话对象。在 更新游戏服务器上的对战数据 (p. 44) 中了解更多信息。
4. 关闭游戏会话的自动回填。 您可以选择在单个游戏会话的任一时刻停止自动回填。要停止自动回填，请将代码添加到您的游戏客户端或游戏服务器，以进行 GameLift API 调用 StopMatchmaking。 此调用需要票证 ID。使用最新回填请求中的回填票证 ID。您可以从游戏会话对战数据中获取此信息，这些数据会按上一步中所述进行更新。

发送回填请求（从游戏服务器）

您可以直接从托管游戏会话的游戏服务器进程发出对战回填请求。该服务器进程具有有关已连接到游戏的当前玩家及空余玩家位置状态的最新信息。

本主题假定您已构建必需的 FlexMatch 组件并已将对战过程成功地添加到您的游戏服务器和客户端游戏服务。有关设置 FlexMatch 的详细信息，请参阅 FlexMatch 与 GameLift 托管的集成 (p. 8)。

要为您的游戏启用对战回填，请添加以下功能：

- 将对战回填请求发送到对战构建器并跟踪请求的状态。
- 更新游戏会话的对战信息。请参阅 更新游戏服务器上的对战数据 (p. 44)。

与其他服务器功能一样，游戏服务器使用 Amazon GameLift 服务器软件开发工具包。此开发工具包在 C++ 和 C# 中可用。

要从您的游戏服务器提出对战回填请求，请完成以下任务。

1. 触发对战回填请求。 通常，每当已对战的游戏具有一个或多个空余玩家位置时，您就会想要发出回填请求。您可能希望将回填请求绑定到特定情况，例如填充关键人物角色或平衡团队。您还有可能想要基于游戏会话的时长限制回填活动。
2. 创建回填请求。 添加代码以创建对战回填请求并将其发送到 FlexMatch 对战构建器。回填请求是使用以下服务器 APIs 处理的：

   - StartMatchBackfill()
   - StopMatchBackfill()

   要创建回填请求，请使用以下信息调用 `StartMatchBackfill`。要取消回填请求，请使用回填请求的票证 ID 调用 `StopMatchBackfill`。

   - 票证 ID — 提供对战票证 ID (或者选择自动生成该 ID)。您可以使用相同的机制将票证 IDs 分配给对战请求和回填请求。以相同方式处理对战和回填的票证。

- 对战构建器 — 确定要用于回填请求的对战构建器。通常，您想要使用与用于创建原始对战的对战构建器相同的构建器。此请求需要对战配置 ARN。此信息存储在游戏会话对象（GameSession）中，该对象在激活游戏会话时由 Amazon GameLift 提供给服务器进程。对战配置 ARN 包含在 `MatchmakerData` 属性中。
- 游戏会话 ARN — 确定要回填的游戏会话。您可以通过调用服务器 API GetGameSessionId（）来获取游戏会话 ARN。对战过程期间，新请求的票证不具有游戏会话 ID，而回填请求的票证则具有。提供游戏会话 ID 是用于区分新对战票证和回填票证的一种方式。
- 玩家数据 — 包含您正在回填的游戏会话中所有当前玩家 (Player) 的玩家信息。此信息让对战构建器能够为当前游戏会话中的玩家找到可能的最佳玩家匹配。如果您的游戏服务器已准确报告玩家连接状态，则您应能够获取此数据，如下所示：
  1. 托管游戏会话的服务器进程应具有玩家当前已连接到游戏会话的最新信息。
  2. 要获取玩家 IDs、属性和团队分配，请从游戏会话对象（GameSession）、`MatchmakerData` 属性中提取玩家数据（请参阅使用对战构建器数据 (p. 40)）。对战构建器数据包含已与游戏会话匹配的所有玩家，因此您将需要提取仅当前连接的玩家的玩家数据。
  3. 对于玩家延迟，如果对战构建器调用延迟数据，则从所有当前玩家中收集新的延迟值并将其包含在每个 `Player` 对象中。如果忽略延迟数据而且对战构建器具有延迟规则，则该请求将不会匹配成功。回填请求仅需要游戏会话当前所在的区域的延迟数据。您可以从 `GameSession` 对象的 `GameSessionId` 属性中获取游戏会话的区域；此值是一个 ARN，其中包含了区域。

3. 跟踪回填请求的状态。 Amazon GameLift 使用服务器开发工具包回调函数 `onUpdateGameSession` 针对回填请求的状态更新您的游戏服务器（请参阅初始化服务器进程）。在—处添加代码以处理状态消息—以及由于回填请求成功而更新的游戏会话对象更新游戏服务器上的对战数据 (p. 44)。

   对战构建器在一个游戏会话中一次只能处理一个对战回填请求。如果您需要取消请求，请调用 StopMatchBackfill（）。如果您需要更改请求，请调用 `StopMatchBackfill`，然后提交更新的请求。

发送回填请求（从客户端服务）

作为从游戏服务器发送回填请求的替代方案，您可能希望从客户端游戏服务发送这些请求。要使用此选项，客户端服务必须有权访问有关游戏会话活动和玩家连接的最新数据；如果您的游戏使用会话目录服务，这可能是很好的选择。

本主题假定您已构建必需的 FlexMatch 组件并已将对战过程成功地添加到您的游戏服务器和客户端游戏服务。有关设置 FlexMatch 的详细信息，请参阅 FlexMatch 与 GameLift 托管的集成 (p. 8)。

要为您的游戏启用对战回填，请添加以下功能：

- 将对战回填请求发送到对战构建器并跟踪请求的状态。
- 更新游戏会话的对战信息。请参阅 更新游戏服务器上的对战数据 (p. 44)

与其他客户端功能一样，客户端游戏服务将结合使用 AWS 软件开发工具包与 Amazon GameLift API。C++、C# 和许多其他语言都提供此软件开发工具包。有关客户端 APIs 的常规说明，请参阅 Amazon GameLift 服务 API 参考，其中介绍了 Amazon GameLift 相关操作的低级别服务 API，并提供特定于语言的参考指南链接。

要设置客户端游戏服务以回填对战的游戏，请完成以下任务。

1. 触发回填请求。 通常，每当已对战的游戏具有一个或多个空余玩家位置时游戏都会启动回填请求。您可能希望将回填请求绑定到特定情况，例如填充关键人物角色或平衡团队。您还有可能想要基于游戏会话的时长限制回填。无论您对触发器使用什么，至少您将需要以下信息。您可以通过使用游戏会话 ID 调用 GameSession，从游戏会话对象（DescribeGameSessions）获取此信息。

   - 当前空余玩家位置的数量。此值可通过游戏会话的最大玩家限制和当前玩家数量计算得出。当前玩家数量会在您的游戏服务器每次连接 Amazon GameLift 服务时进行更新以验证新的玩家连接或报告断开连接的玩家。

- 创建策略。此设置指示游戏会话当前是否接受新玩家。

游戏会话对象包含其他可能有用的信息，包括游戏会话开始时间、自定义游戏属性和对战构建器数据。

2. 创建回填请求。 添加代码以创建对战回填请求并将其发送到 FlexMatch 对战构建器。回填请求是使用以下客户端 APIs 处理的：

   - StartMatchBackfill
   - StopMatchmaking

   要创建回填请求，请使用以下信息调用 `StartMatchBackfill`。回填请求类似于对战请求 (请参阅请求玩家对战 (p. 37))，但还识别现有游戏会话。要取消回填请求，请使用回填请求的票证 ID 调用 `StopMatchmaking`。

   - 票证 ID — 提供对战票证 ID (或者选择自动生成该 ID)。您可以使用相同的机制将票证 IDs 分配给对战请求和回填请求。以相同方式处理对战和回填的票证。
   - 对战构建器 — 识别要使用的对战配置的名称。通常，您想要使用与用于创建原始对战的回填的对战构建器相同的构建器。此信息位于对战配置 ARN 下的游戏会话对象（GameSession）、`MatchmakerData` 属性中。名称值是紧接在“matchmakingconfiguration/”之后的字符串。（例如，在 ARN 值“arn:aws-cn:gamelift:us-west-2:111122223333:matchmakingconfiguration/MM-4v4”中，对战配置名称为“MM-4v4”。）
   - 游戏会话 ARN — 指定要回填的游戏会话。使用游戏会话对象中的 `GameSessionId` 属性；此 ID 使用您所需的 ARN 值。回填请求的对战票证（MatchmakingTicket）在处理期间将有游戏会话 ID；新对战请求的票证在对战游戏前不会获取游戏会话 ID；游戏会话 ID 的存在是用于区分新对战票证和回填票证的一种方式。
   - 玩家数据 — 包含您正在回填的游戏会话中所有当前玩家 (Player) 的玩家信息。此信息让对战构建器能够为当前游戏会话中的玩家找到可能的最佳玩家匹配。如果您的游戏服务器已准确报告玩家连接状态，则您应能够获取此数据，如下所示：

     1. 使用游戏会话 ID 调用 DescribePlayerSessions（ ），以发现当前已连接到游戏会话的所有玩家。每个玩家会话包括一个玩家 ID。您可以添加状态筛选器以仅检索活动的玩家会话。
     2. 从游戏会话对象（GameSession）、`MatchmakerData` 属性中提取玩家数据（请参阅使用对战构建器数据 (p. 40)）。使用在上一步中获取的玩家 IDs 来仅获取当前连接玩家的数据。由于玩家退出时不会更新对战构建器数据，因此您仅需要提取当前玩家的数据。
     3. 对于玩家延迟，如果对战构建器调用延迟数据，请从所有当前玩家中收集新的延迟值并将其包含在 `Player` 对象中。如果忽略延迟数据而且对战构建器具有延迟规则，则该请求将不会匹配成功。回填请求仅需要游戏会话当前所在的区域的延迟数据。您可以从 `GameSession` 对象的 `GameSessionId` 属性中获取游戏会话的区域；此值是一个 ARN，其中包含了区域。

3. 跟踪回填请求的状态。 添加代码以侦听对战票证状态更新。您可以使用设置的机制利用事件通知 (首选) 或轮询跟踪新对战请求的票证 (请参阅跟踪对战事件 (p. 38))。尽管您无需使用回填请求触发玩家接受活动，而且玩家信息已在游戏服务器上更新，但仍需要监控票证状态以处理请求失败和重新提交。

   对战构建器在一个游戏会话中一次只能处理一个对战回填请求。如果您需要取消请求，请调用 StopMatchmaking。 如果您需要更改请求，请调用 `StopMatchmaking`，然后提交更新的请求。

   在对战回填请求成功后，您的游戏服务器会收到更新的 `GameSession` 对象并处理将新玩家加入游戏会话中所需的任务。请在更新游戏服务器上的对战数据 (p. 44)上查看更多信息。

更新游戏服务器上的对战数据

无论在您的游戏中如何启动对战回填请求，您的游戏服务器都必须能够处理由于对战回填请求而导致 Amazon GameLift 提供的游戏会话更新。

当 Amazon GameLift 成功完成对战回填请求时 — 它使用回调函数 — 调用您的游戏服务器。`onUpdateGameSession` 此调用具有三个输入参数：对战回填票证 ID、状态消息和包含最新对战数据（包括玩家信息）的 GameSession 对象。您需要将以下代码添加到游戏服务器以作为您的游戏服务器集成的一部分：

1. 实现 `onUpdateGameSession` 函数。此函数必须能够处理以下状态消息 (`updateReason`)：
   - MATCHMAKING_DATA_UPDATED – 新玩家已与游戏会话成功匹配。`GameSession` 对象包含更新的对战构建器数据，包括有关现有玩家和新匹配的玩家的玩家数据。
   - BACKFILL_FAILED – 对战回填尝试由于内部错误而失败。`GameSession` 对象保持不变。
   - BACKFILL_TIMED_OUT – 对战构建器未能在时间限制内找到回填对战。`GameSession` 对象保持不变。
   - BACKFILL_CANCELLED – 对战回填请求已通过调用 StopMatchmaking（客户端）或 StopMatchBackfill（服务器）而被取消。`GameSession` 对象保持不变。
2. 对于成功的回填对战，请使用更新的对战构建器数据来在新玩家连接到游戏会话时进行处理。至少，您将需要使用新玩家的团队任务以及要让玩家在游戏中开始所需的其他玩家属性。
3. 在游戏服务器对服务器开发工具包操作 ProcessReady（ ）的调用中，添加 `onUpdateGameSession` 回调方法名称作为进程参数。

使用 Amazon CloudWatch 监控 FlexMatch

使用 Amazon CloudWatch 指标扩展...。

GameLift FlexMatch 参考

本部分包含与 GameLift FlexMatch 进行对战的参考文档。

主题

GameLift FlexMatch API 参考（AWS 开发工具包）

本主题提供了适用于 GameLift FlexMatch 的基于任务的 API 操作列表。GameLift 服务 API 已打包到 FlexMatch 命名空间的 AWS 开发工具包中。`aws.gamelift`下载 AWS SDK或查看 Amazon GameLift API 参考文档。

GameLift FlexMatch 提供用于托管于 GameLift 托管解决方案（包括用于自定义游戏服务器的托管或 实时服务器 的托管，在具有 Amazon EC2 的 GameLift FleetIQ 上托管）的游戏以及其他托管系统（如对等系统、本地或云计算基元）的对战服务。有关其他 托管选项的更多信息，请参阅 GameLift 开发人员指南。GameLift

设置对战规则和流程

调用这些操作以创建 FlexMatch 对战构建器，为游戏配置对战过程并定义用于创建对战和团队的一组自定义规则。

对战配置

- CreateMatchmakingConfiguration – 创建对战配置，其中包含用于评估玩家组和构建玩家团队的说明。使用 GameLift 进行托管时，还要指定如何为对战游戏创建新游戏会话。
- DescribeMatchmakingConfigurations 检索定义 – 区域的对战配置。GameLift
- UpdateMatchmakingConfiguration – 更改对战配置队列的设置。
- DeleteMatchmakingConfiguration – 从区域中删除对战配置。

对战规则集

- CreateMatchmakingRuleSet – 创建一组在搜索玩家匹配时使用的规则。
- DescribeMatchmakingRuleSets 检索在 – 区域中定义的对战规则集。GameLift
- ValidateMatchmakingRuleSet – 验证一组对战规则的语法。
- DeleteMatchmakingRuleSet – 从区域中删除对战规则集。

为玩家请求对战游戏

从游戏客户端服务调用这些操作来管理玩家对战请求。

- StartMatchmaking – 为一个玩家或要玩同一对战的组请求对战游戏。

- DescribeMatchmaking – 获取对战请求的详细信息，包括状态。
- AcceptMatch – 对于需要玩家接受的对战游戏，在玩家接受建议的对战游戏时通知 GameLift。
- StopMatchmaking – 取消对战请求。
- StartMatchBackfill - 请求其他玩家对战以填充现有游戏会话中的空位置。

可用编程语言

AWS SDK和 Amazon GameLift 提供以下语言版本。有关开发环境的支持详情，请参阅每种语言的文档。

- C++ (开发工具包文档) (Amazon GameLift)
- Java (开发工具包文档) (Amazon GameLift)
- .NET (开发工具包文档) (Amazon GameLift)
- Go (开发工具包文档) (Amazon GameLift)
- Python (开发工具包文档) (Amazon GameLift)
- Ruby (开发工具包文档) (Amazon GameLift)
- PHP (开发工具包文档) (Amazon GameLift)
- JavaScript/Node.js (开发工具包文档) (Amazon GameLift)

FlexMatch 规则集架构

FlexMatch 规则集对小对战规则和大对战规则使用标准模式。在创建游戏的对战规则时，将此架构与FlexMatch 规则语言 (p. 54)中描述的规则语言一起使用。

了解有关创建 FlexMatch 规则的更多信息：

- 设计 FlexMatch 规则集 (p. 13)
- 创建对战规则集 (p. 20)
- FlexMatch 规则集示例 (p. 22)
- FlexMatch 规则语言 (p. 54)

小型对战的规则集架构

以下架构记录了用于构建最多 40 位玩家的对战的规则集的所有可能的属性和允许值。

```
{
    "name": "string",
    "ruleLanguageVersion": "1.0",
    "playerAttributes":[{
         "name": "string,
         "type": <"string", "number", "string_list", "string_number_map">,
         "default": "string"
    }],
    "algorithm": {
        "strategy": "exhaustiveSearch",
        "batchingPreference": <"random", "sorted">,
        "sortByAttributes": [ "string" ],
        "expansionAgeSelection": <"newest", "oldest">,
        "backfillPriority": <"normal", "low", "high">
    },
    "teams": [{
```

```
        "name": "string",
        "maxPlayers": number,
        "minPlayers": number,
        "quantity": integer
    }],
    "rules": [{
        "type": "distance",
        "name": "string",
        "description": "string",
        "measurements": "string",
        "referenceValue": number,
        "maxDistance": number,
        "minDistance": number,
        "partyAggregation": <"avg", "min", "max">
        },{
        "type": "comparison",
        "name": "string",
        "description": "string",
        "measurements": "string",
        "referenceValue": number,
        "operation": <"<", "<=", "=", "!=", ">", ">=">,
        "partyAggregation": <"avg", "min", "max">
        },{
        "type": "collection",
        "name": "string",
        "description": "string",
        "measurements": "string",
        "referenceValue": number,
        "operation": <"intersection", "contains", "reference_intersection_count">,
        "maxCount": number,
        "minCount": number,
        "partyAggregation": <"union", "intersection">
        },{
        "type": "latency",
        "name": "string",
        "description": "string",
        "maxLatency": number,
        "maxDistance": number,
        "distanceReference": number,
        "partyAggregation": <"avg", "min", "max">
        },{
        "type": "distanceSort",
        "name": "string",
        "description": "string",
        "sortDirection": <"ascending", "descending">,
        "sortAttribute": "string",
        "mapKey": <"minValue", "maxValue">,
        "partyAggregation": <"avg", "min", "max">
        },{
        "type": "absoluteSort",
        "name": "string",
        "description": "string",
        "sortDirection": <"ascending", "descending">,
        "sortAttribute": "string",
        "mapKey": <"minValue", "maxValue">,
        "partyAggregation": <"avg", "min", "max">
    }],
    "expansions": [{
        "target": "string",
        "steps": [{
            "waitTimeSeconds": number,
            "value": number
        }, {
            "waitTimeSeconds": number,
            "value": number
        }]
```

```
    }]
}
```

大型对战的规则集架构

以下架构记录了用于构建超过 40 位玩家的对战的规则集的所有可能的属性和允许值。如果规则集中所有团队的总 `maxPlayers` 值超过 40，FlexMatch 将在大型对战指南下处理使用此规则集的对战请求。

```
{
    "name": "string",
    "ruleLanguageVersion": "1.0",
    "playerAttributes":[{
         "name": "string,
         "type": <"string", "number", "string_list", "string_number_map">,
         "default": "string"
    }],
    "algorithm": {
        "strategy": "balanced",
        "batchingPreference": <"largestPopulation", "fastestRegion">,
        "balancedAttribute": "string",
        "expansionAgeSelection": <"newest", "oldest">,
        "backfillPriority": <"normal", "low", "high">
    },
    "teams": [{
        "name": "string",
        "maxPlayers": number,
        "minPlayers": number,
        "quantity": integer
    }],
    "rules": [{
        "type": "latency",
        "name": "string",
        "description": "string",
        "maxLatency": number,
        "partyAggregation": <"avg", "min", "max">
    }],
    "expansions": [{
        "target": "string",
        "steps": [{
            "waitTimeSeconds": number,
            "value": number
        }, {
            "waitTimeSeconds": number,
            "value": number
        }]
    }]
}
```

规则集架构属性定义

此部分定义规则集架构中的每个属性。有关创建规则集的更多帮助，请参阅设计 FlexMatch 规则集 (p. 13)。

`name`

规则集的描述标签。此值与分配给 GameLift MatchmakingRuleSet 资源的名称不关联。该值包含在描述已完成对战游戏的对战数据中，但未被任何 GameLift 进程使用。

允许的值：字符串

必需？ 否

ruleLanguageVersion

正在使用的 FlexMatch 属性表达式语言的版本。

允许的值：“1.0”

必需？ 是

playerAttributes

包含在对战请求中并在对战过程中使用的玩家数据的集合。您也可以在此处声明属性，以便将玩家数据包含在传递到游戏服务器的对战数据中，即使该数据在对战过程中未使用。

必需？ 否

name

对战构建器要使用的玩家属性的唯一名称。此名称必须与对战请求中引用的玩家属性名称匹配。

允许的值：字符串

必需？ 是

type

玩家属性值的数据类型。

允许的值：“string”、“number”、“string_list”、“string_number_map”

必需？ 是

default

对战请求未为玩家提供对战请求时使用的默认值。

允许的值：玩家属性允许的任何值。

必需？ 否

algorithm

用于自定义对战过程的可选配置设置。

必需？ 否

strategy

构建对战游戏时使用的方法。如果未设置此属性，则默认行为是“exhativeSearch”。

允许的值：

- “exhaustiveSearch”– 标准匹配方法。FlexMatch 通过评估池中基于一组自定义匹配规则的其他票证，围绕一个批次中的最旧票证形成匹配项。此策略用于 40 位玩家或更少玩家的对战游戏。使用此策略时，`batchingPreference` 应设置为“随机”或“已排序”。
- “平衡”– 方法，经优化可快速形成大型对战游戏。此策略仅用于 41 到 200 位玩家的对战游戏。它通过对票证池进行预排序、构建潜在对战并将玩家分配到团队，然后使用指定的玩家属性平衡对战游戏中的每个团队来构建对战游戏。例如，此策略可用于相等对战游戏中所有团队的平均技能级别。在使用此策略时，必须设置 `balancedAttribute`，并且 `batchingPreference` 应设置为“largestPopulation”或“fastestRegion”。大部分自定义规则类型无法识别为此策略。

必需？ 是

batchingPreference

对对对战构建的票证进行分组之前要使用的预排序方法。对票证池进行预排序时，会导致根据特定特征对票证进行批处理，这往往会增加最终对战游戏中玩家之间的一致性。

允许的值：

- “random”– 仅当 `strategy` = "exhausiveSearch" 时才有效。不会执行预排序；池中的票证会随机分批。这是详尽搜索策略的默认行为。
- “sorted”– 仅当 `strategy` = "exhaustiveSearch" 时才有效。票证池根据 `sortbyAttributes` 中列出的玩家属性进行预排序。
- “largestPopulation”– 只在 `strategy` =“balanced”时有效。票证池按玩家报告可接受的延迟级别的区域预先排序。这是平衡策略的默认行为。
- “fastestRegion”– 只在 `strategy` =“balanced”时有效。票证池按玩家报告其最低延迟级别的区域预先排序。生成的对战游戏需要更长的时间才能完成，但所有玩家的延迟往往较低。

必需？ 是

`balancedAttribute`

使用平衡策略构建大型对战游戏时使用的玩家属性的名称。

允许的值：在 `playerAttributes` 中声明并且 `type` =“number”的任何属性。

必需？ 是，如果 `strategy` =“balanced”。

`sortByAttributes`

在批处理之前对票证池进行预排序时使用的玩家属性列表。此属性仅在使用详尽搜索策略进行预排序时使用。属性列表的顺序确定排序顺序。FlexMatch 对 Alpha 和数值使用标准排序约定。

允许的值：在 `playerAttributes` 中声明的任何属性。

必需？ 是，如果 `batchingPreference` = "sorted"。

`backfillPriority`

用于匹配回填票证的优先级方法。此属性确定 FlexMatch 何时处理批处理中的回填票证。仅当使用详尽搜索策略进行预排序时使用。如果未设置此属性，则默认行为是“正常”。

允许的值：

- “正常”– 构建对战游戏时，不考虑票证的请求类型（回填或新对战游戏）。
- “高”– 票证批处理按请求类型（然后按期限）排序，FlexMatch 首先尝试匹配回填票证。
- “低”– 票证批处理按请求类型（然后按期限）进行排序，FlexMatch 首先尝试匹配非回填票证。

必需？ 否

`expansionAgeSelection`

计算对战规则扩展等待时间的方法。扩展用于在经过一定时间量后未完成对战游戏时放宽对战游戏要求。等待时间是根据已在部分填充的对战游戏中的票证的存在时间计算的。如果未设置此属性，则默认行为是“新”。

允许的值：

- “最新”– 根据在部分完成的对战游戏中具有最新创建时间戳的票证来计算扩展等待时间。扩展触发的速度往往更慢，因为一个较新的票证可以重新启动等待时间时钟。
- “最早”– 根据对战游戏中具有最早创建时间戳的票证计算扩展等待时间。扩展往往能更快地触发。

必需？ 否

`teams`

对战游戏中的团队配置。为每个团队提供团队名称和规模范围。一个规则集必须定义至少一个团队。

`name`

团队的唯一名称。可以在规则和扩展中引用团队名称。在成功进行对战游戏时，玩家按对战数据中的团队名称分配。

允许的值：字符串

必需？ 是

maxPlayers

可以分配给团队的玩家数量上限。

允许的值：数字

必需？ 是

minPlayers

在对战游戏可行之前必须分配给团队的最小玩家数量。

允许的值：数字

必需？ 是

quantity

要在对战游戏中创建的此类型的团队的数量。数量大于 1 的团队使用附加编号（“Red_1”、“Red_2”等）指定。如果未设置此属性，则默认值为“1”。

允许的值：数字

必需？ 否

rules

定义如何为对战评估玩家的规则语句集合。

必需？ 否

name

规则的唯一名称。规则集中的所有规则必须具有唯一的名称。规则名称在跟踪与规则相关的活动的事件日志和指标中引用。

允许的值：字符串

必需？ 是

description

规则的文本描述。此信息不用于对战过程。

允许的值：字符串

必需？ 否

type

规则语句的类型。每个规则类型都有必须设置的其他属性。有关如何完全定义每种类型的规则的更多详细信息，请参阅FlexMatch 规则语言 (p. 54)。

允许的值：

- “distance”– 测量数字值之间的距离。
- “比较”– 比较两个值。
- “集合”– 评估集合中的值，如集合的玩家属性或一组适用于多个玩家的值。
- “延迟”– 评估为对战请求报告的区域延迟数据。
- “distanceSort”– 根据具有数值的指定玩家属性与批次中最旧票证的比较方式，分批对票证进行排序的显式排序方法。
- “absoluteSort”– 根据指定的玩家属性是否与批次中最旧的票证进行比较，对批次中的票证进行排序的显式排序方法。

必需？ 是

expansions

在无法完成对战游戏时随着时间推移放宽对战游戏要求的规则。将扩展设置为一系列步骤，这些步骤将逐渐应用以使匹配项更易于查找。默认情况下，FlexMatch 根据添加到对战游戏的最新票证的期限计算等待时间。您可以使用算法属性 `expansionAgeSelection` 更改计算扩展等待时间的方式。

扩展等待时间是绝对值，因此每个步骤的等待时间应比上一步长。例如，要计划一系列逐步扩展，您可以使用 30 秒、40 秒和 50 秒的等待时间。等待时间不能超过对战请求允许的最大时间（在对战配置中设置）。

必需？ 否

target

要放宽的规则集元素。您可以放宽团队大小属性或任何规则语句属性。语法为“<component name>rule/team name&gt；”。<property name> 例如，要更改团队最小规模，请执行以下操作：`teams[Red, Yellow].minPlayers`。 要在名为“minSkill”的比较规则语句中更改最低技能要求，请执行以下操作：`rules[minSkill].referenceValue`。

必需？ 是

steps

waitTimeSeconds

在为目标规则集元素应用新值之前等待的时间长度（以秒为单位）。

必需？ 是

value

目标规则集元素的新值。

FlexMatch 规则语言

为 FlexMatch 规则集编写规则时，使用下列属性表达式语法。

了解有关创建 FlexMatch 规则的更多信息：

- 构建 FlexMatch 规则集 (p. 13)
- 创建对战规则集 (p. 20)
- FlexMatch 规则集示例 (p. 22)

规则类型

FlexMatch 支持以下规则类型。每个规则类型都需要一组属性，此处将对其进行介绍。

距离规则 (**distance**)

距离规则用于度量两个数字值之间的差值，例如技能级别之间的差距。例如，距离规则可能要求所有玩家都在两个技能级别内。

距离规则属性

- **measurements** – 为其度量距离的玩家属性值；必须为数字值。
- **referenceValue** – 用于对照潜在对战游戏度量距离的数字值。
- **minDistance/maxDistance** – 为了成功进行对战游戏而允许的最大或最小距离值。
- **partyAggregation** 处理具有多个玩家的票证的方法（广告）。–有效选项是使用票证玩家的最小值（`min`）、最大值（`max`）或平均值（`avg`）。默认为 `avg`。

比较规则 (**comparison**)

比较规则对照其他值测量玩家属性值。有两种类型的比较规则。第一种类型将属性值与提供的参考值进行比较。指定参考值和有效的比较操作。例如，规则可能要求匹配的玩家拥有 24 或更高的技能水平。第二种类型将比较团队中玩家的玩家属性值，或进行对战以确定所有玩家是否没有玩家具有相同的属性值。使用此类型的比较规则，请省略引用值并指定等于 （“=”） 或不等于 （“!=”） 操作。例如，规则可能要求对战游戏中的所有玩家选择相同的游戏地图，或要求团队中没有玩家选择相同的角色。

比较规则属性

- **measurements** – 要比较的玩家属性值。
- **referenceValue** – 用于对照潜在对战游戏评估测量值的值。
- **operation** – 如何评估测量值。有效操作包括：`<`, `<=`, `=`, `!=`, `>`, `>=`。
- **partyAggregation** 处理具有多个玩家的票证的方法 （广告）。–有效选项是使用票证玩家的最小值 （`min`）、最大值 （`max`） 或平均值 （`avg`）。默认为 `avg`。

收集规则 (**collection**)

集合规则评估玩家属性值的集合。一个集合可以包含多个玩家的属性值和/或采用集合格式 (字符串列表) 的一个玩家属性。例如，某个集合规则可能会查看玩家在团队中选择的角色，并且要求该团队具有至少一个特定角色。

收集规则属性

- **measurements** – 要评估的玩家属性值的集合。属性值必须采用字符串列表形式。
- **referenceValue** – 值（或值集合），用于对照潜在对战游戏评估测量值。
- **operation** – 如何评估测量值集合。有效操作包括：
  - `intersection` 计量所有玩家集合中的通用的值的数量。请参阅MapOverlap中的 示例 4：使用显式排序查找最佳匹配 (p. 27) 规则。
  - `contains` 计量包含特定参考值的玩家属性集合的数量。请参阅OverallMedicLimit中的 示例 3：设置团队级要求和延迟限制 (p. 25) 规则。
  - `reference_intersection_count` 度量玩家属性集合与参考值集合之间的交集。在 `measurements` 中定义的每个玩家属性集合（字符串列表）都会单独对照引用集合进行评估。此操作可用于在不同的玩家属性之间进行评估。请参阅OpponentMatch中的 示例 5：查找多个玩家属性的交集 (p. 28) 规则。
- **minCount/maxCount** – 为了成功进行对战游戏而允许的最大或最小计数值。
- **partyAggregation** 处理具有多个玩家的票证的方法 （广告）。–有效选项是使用票证玩家的 `union` 或 `intersection` 值。默认为 `union`。

延迟规则 (**latency**)

延迟规则按区域评估玩家延迟报告 - 例如，规则可能要求匹配的玩家必须在低于最大限制的同一区域中具有延迟。这是大型对战游戏唯一允许的规则类型，而设置 `maxLatency` 是唯一允许的属性。

延迟规则属性

- **maxLatency** – 区域的最高可接受延迟值。对于每个票证，忽略超过此延迟的所有区域。
- **maxDistance** – 每个票证的延迟与距离参考值之间的最大差值。
- **distanceReference** – 与 `maxDistance` 配合使用。 要对照成功对战游戏度量距离的数字值。对于延迟，此值是多个玩家的延迟值的聚合。有效选项为玩家延迟的最小值 (`min`) 或平均值 (`avg`)。(请参阅属性表达式部分。)
- **partyAggregation** 处理具有多个玩家的票证的方法 （广告）。–有效选项是使用票证玩家的最小值 （`min`）、最大值 （`max`） 或平均值 （`avg`）。默认为 `avg`。

距离排序规则 (**distanceSort**)

距离排序是一个显式排序选项，该选项指示对战构建器根据玩家属性对一批对战票证进行排序。距离排序规则根据与最旧票证的距离评估票证。

距离排序规则属性

- **sortDirection** – 有关对战票证排序的说明。有效选项是 `ascending` 或 `descending`。

- **sortAttribute** – 用作玩家排序依据的玩家属性。
- **mapKey** – 如何对玩家的地图属性进行评估。有效选项包括：
  - minValue：对于最旧的票证，找到具有最低值的密钥。
  - maxValue：对于最旧的票证，找到具有最高值的密钥。
- **partyAggregation** 处理具有多个玩家的票证的方法（广告）。–有效选项是使用票证玩家的最小值（min）、最大值（max）或平均值（avg）。默认为 avg。

绝对排序规则 (**absoluteSort**)

绝对排序是一个显式排序选项，该选项指示对战构建器根据玩家属性对一批对战票证进行排序。绝对排序根据其玩家属性是否与最旧的票证匹配来评估对战票证。

绝对排序规则属性

- **sortDirection** – 有关对战票证排序的说明。有效选项是 ascending 或 descending。
- **sortAttribute** – 用作玩家排序依据的玩家属性。
- **mapKey** – 如何对玩家的地图属性进行评估。有效选项包括：
  - minValue：对于最旧的票证，找到具有最低值的密钥。
  - maxValue：对于最旧的票证，找到具有最高值的密钥。
- **partyAggregation** 处理具有多个玩家的票证的方法（广告）。–有效选项是使用票证玩家的最小值（min）、最大值（max）或平均值（avg）。默认为 avg。

属性表达式

属性表达式可用于定义与对战相关的某些属性。它们允许您在定义属性值时使用计算和逻辑。属性表达式通常生成以下两种形式之一：

- 单个玩家数据。
- 单个玩家数据的计算集合。

有效属性表达式标识单个玩家、团队或对战游戏的特定值。以下部分表达式说明了如何标识团队和玩家：

目标	Input	意义	输出
标识对战游戏的特定团队：	teams[red]	红队	团队
标识对战游戏的所有团队：	teams[*]	所有团队	List<Team>
标识特定团队中的玩家：	team[red].players	红队中的玩家	List<Player>
标识对战游戏的玩家：	team[*].players	对战游戏的玩家 (按团队分组)	List<List<Player>>

下表给出基于之前示例构建的部分有效属性表达式：

表达式	意义	结果类型
teams[red].players[playerid]	红队所有玩家的玩家IDs	List<string>
teams[red].players.attribute[skill]	红队所有玩家的“技能”属性	List<number>

表达式	意义	结果类型
teams[*].players.attributes[skill]	对战游戏的所有玩家的“技能”属性 (按团队分组)	List<List<number>>

属性表达式可用于使用以下函数或组合函数来聚合团队数据：

聚合	Input	意义	输出
min	List<number>	获取列表中所有数字的最小值。	number
max	List<number>	获取列表中所有数字的最大值。	number
avg	List<number>	获取列表中所有数字的平均值。	number
median	List<number>	获取列表中所有数字的中值。	number
sum	List<number>	获取列表中所有数字的总和。	number
count	List<?>	获取列表中的元素数量。	number
stddev	List<number>	获取列表中所有数字的标准差。	number
flatten	List<List<?>>	将嵌套列表的集合变成包含所有元素的单个列表。	List<?>
set_intersection	List<List<string>>	获取在集合的所有字符串列表中找到的字符串列表。	List<string>
以上全部	List<List<?>>	对嵌套列表的所有操作会对每个子列表执行一次以生成结果列表。	List<?>

下表给出使用聚合函数的部分有效属性表达式：

表达式	意义	结果类型
flatten(teams[*].players.attributes[skill])	对战游戏中的所有玩家的“技能”属性 (未分组)	List<number>
avg(teams[red].players.attributes[skill])	红队玩家的平均技能	number
avg (teams[*].players.attributes[Skill])	对战游戏中的每个团队的平均技能	List<number>
avg(flatten(teams[*].players.attributes[skill]))	对战游戏中的所有玩家的平均技能级别。该表达式获取玩家技能的展	number

表达式	意义	结果类型
	开列表，然后计算它们的平均值。	
count(teams[red].players)	红队的玩家数量	number
count (teams[*].players)	对战游戏中的每个团队的玩家数量	List<number>
max(avg(teams[*].players.attributes[skill]))	对战游戏中的最高团队技能级别	number

FlexMatch 对战事件

Amazon GameLift 发出与对战票证处理相关的事件。此处列出的所有事件都能发布到 Amazon SNS 主题。这些事件也会发送到 Amazon CloudWatch Events。有关使用对战事件的更多详细信息，请参阅设置 FlexMatch 事件通知 (p. 34)。有关对战票证状态的更多信息，请参阅 服务 API 参考中的 MatchmakingTicketAmazon GameLift。

MatchmakingSearching

票证已输入到对战中。这包括新的请求和失败的建议对战游戏所包含的请求。

资源： ConfigurationArn

详细信息：类型、票证、estimatedWaitMillis、gameSessionInfo

Example

```
{
  "version": "0",
  "id": "cc3d3ebe-1d90-48f8-b268-c96655b8f013",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-08T21:15:36.421Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-08T21:15:35.676Z",
        "players": [
          {
            "playerId": "player-1"
          }
        ]
      }
    ],
    "estimatedWaitMillis": "NOT_AVAILABLE",
    "type": "MatchmakingSearching",
    "gameSessionInfo": {
      "players": [
        {
```

```
          "playerId": "player-1"
        }
      ]
    }
  }
}
```

PotentialMatchCreated

潜在的对战游戏已创建。这是对所有新潜在对战游戏发出的，不论是否需要接受。

资源： ConfigurationArn

详细信息：类型、票
证、acceptanceTimeout、acceptanceRequired、ruleEvaluationMetrics、gameSessionInfo、matchId

Example

```
{
  "version": "0",
  "id": "fce8633f-aea3-45bc-aeba-99d639cad2d4",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-08T21:17:41.178Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-08T21:15:35.676Z",
        "players": [
          {
            "playerId": "player-1",
            "team": "red"
          }
        ]
      },
      {
        "ticketId": "ticket-2",
        "startTime": "2017-08-08T21:17:40.657Z",
        "players": [
          {
            "playerId": "player-2",
            "team": "blue"
          }
        ]
      }
    ],
    "acceptanceTimeout": 600,
    "ruleEvaluationMetrics": [
      {
        "ruleName": "EvenSkill",
        "passedCount": 3,
        "failedCount": 0
      },
      {
        "ruleName": "EvenTeams",
```

```
        "passedCount": 3,
        "failedCount": 0
      },
      {
        "ruleName": "FastConnection",
        "passedCount": 3,
        "failedCount": 0
      },
      {
        "ruleName": "NoobSegregation",
        "passedCount": 3,
        "failedCount": 0
      }
    ],
    "acceptanceRequired": true,
    "type": "PotentialMatchCreated",
    "gameSessionInfo": {
      "players": [
        {
          "playerId": "player-1",
          "team": "red"
        },
        {
          "playerId": "player-2",
          "team": "blue"
        }
      ]
    },
    "matchId": "3faf26ac-f06e-43e5-8d86-08feff26f692"
  }
}
```

AcceptMatch

玩家已接受潜在的对战游戏。此事件包含对战游戏中每个玩家的当前接受状态。缺少数据意味着尚未为该玩家调用 AcceptMatch。

资源： ConfigurationArn

详细信息：类型、票证、matchId、gameSessionInfo

Example

```
{
  "version": "0",
  "id": "b3f76d66-c8e5-416a-aa4c-aa1278153edc",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-09T20:04:42.660Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-09T20:01:35.305Z",
        "players": [
          {
```

```
            "playerId": "player-1",
            "team": "red"
          }
        ]
      },
      {
        "ticketId": "ticket-2",
        "startTime": "2017-08-09T20:04:16.637Z",
        "players": [
          {
            "playerId": "player-2",
            "team": "blue",
            "accepted": false
          }
        ]
      }
    ],
    "type": "AcceptMatch",
    "gameSessionInfo": {
      "players": [
        {
          "playerId": "player-1",
          "team": "red"
        },
        {
          "playerId": "player-2",
          "team": "blue",
          "accepted": false
        }
      ]
    },
    "matchId": "848b5f1f-0460-488e-8631-2960934d13e5"
  }
}
```

AcceptMatchCompleted

一旦玩家接受、玩家拒绝或接受超时，对战游戏接受操作即完成。

资源： ConfigurationArn

详细信息：类型、票证、接受、matchId、gameSessionInfo

Example

```
{
  "version": "0",
  "id": "b1990d3d-f737-4d6c-b150-af5ace8c35d3",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-08T20:43:14.621Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-08T20:30:40.972Z",
        "players": [
```

```
          {
            "playerId": "player-1",
            "team": "red"
          }
        ]
      },
      {
        "ticketId": "ticket-2",
        "startTime": "2017-08-08T20:33:14.111Z",
        "players": [
          {
            "playerId": "player-2",
            "team": "blue"
          }
        ]
      }
    ],
    "acceptance": "TimedOut",
    "type": "AcceptMatchCompleted",
    "gameSessionInfo": {
      "players": [
        {
          "playerId": "player-1",
          "team": "red"
        },
        {
          "playerId": "player-2",
          "team": "blue"
        }
      ]
    },
    "matchId": "a0d9bd24-4695-4f12-876f-ea6386dd6dce"
  }
}
```


MatchmakingSucceeded

对战匹配已成功完成并已创建游戏会话。

资源： ConfigurationArn

详细信息：类型、票证、matchId、gameSessionInfo

Example


```
{
  "version": "0",
  "id": "5ccb6523-0566-412d-b63c-1569e00d023d",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-09T19:59:09.159Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-09T19:58:59.277Z",
        "players": [
```

```
          {
            "playerId": "player-1",
            "playerSessionId": "psess-6e7c13cf-10d6-4756-a53f-db7de782ed67",
            "team": "red"
          }
        ]
      },
      {
        "ticketId": "ticket-2",
        "startTime": "2017-08-09T19:59:08.663Z",
        "players": [
          {
            "playerId": "player-2",
            "playerSessionId": "psess-786b342f-9c94-44eb-bb9e-c1de46c472ce",
            "team": "blue"
          }
        ]
      }
    ],
    "type": "MatchmakingSucceeded",
    "gameSessionInfo": {
      "gameSessionArn": "arn:aws-cn:gamelift:us-west-2:123456789012:gamesession/836cf48d-
bcb0-4a2c-bec1-9c456541352a",
      "ipAddress": "192.168.1.1",
      "port": 10777,
      "players": [
        {
          "playerId": "player-1",
          "playerSessionId": "psess-6e7c13cf-10d6-4756-a53f-db7de782ed67",
          "team": "red"
        },
        {
          "playerId": "player-2",
          "playerSessionId": "psess-786b342f-9c94-44eb-bb9e-c1de46c472ce",
          "team": "blue"
        }
      ]
    },
    "matchId": "c0ec1a54-7fec-4b55-8583-76d67adb7754"
  }
}
```

MatchmakingTimedOut

对战票证由于超时而失败。

资源： ConfigurationArn

详细信息：类型、票证、ruleEvaluationMetrics、消息、matchId、gameSessionInfo

Example

```
{
  "version": "0",
  "id": "fe528a7d-46ad-4bdc-96cb-b094b5f6bf56",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-09T20:11:35.598Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
```

```
  ],
  "detail": {
    "reason": "TimedOut",
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-09T20:01:35.305Z",
        "players": [
          {
            "playerId": "player-1",
            "team": "red"
          }
        ]
      }
    ],
    "ruleEvaluationMetrics": [
      {
        "ruleName": "EvenSkill",
        "passedCount": 3,
        "failedCount": 0
      },
      {
        "ruleName": "EvenTeams",
        "passedCount": 3,
        "failedCount": 0
      },
      {
        "ruleName": "FastConnection",
        "passedCount": 3,
        "failedCount": 0
      },
      {
        "ruleName": "NoobSegregation",
        "passedCount": 3,
        "failedCount": 0
      }
    ],
    "type": "MatchmakingTimedOut",
    "message": "Removed from matchmaking due to timing out.",
    "gameSessionInfo": {
      "players": [
        {
          "playerId": "player-1",
          "team": "red"
        }
      ]
    }
  }
}
```


MatchmakingCancelled

对战票证已取消。

资源： ConfigurationArn

详细信息：类型、票证、ruleEvaluationMetrics、消息、matchId、gameSessionInfo

Example


```
{
  "version": "0",
```

```
  "id": "8d6f84da-5e15-4741-8d5c-5ac99091c27f",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-09T20:00:07.843Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
  ],
  "detail": {
    "reason": "Cancelled",
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-09T19:59:26.118Z",
        "players": [
          {
            "playerId": "player-1"
          }
        ]
      }
    ],
    "ruleEvaluationMetrics": [
      {
        "ruleName": "EvenSkill",
        "passedCount": 0,
        "failedCount": 0
      },
      {
        "ruleName": "EvenTeams",
        "passedCount": 0,
        "failedCount": 0
      },
      {
        "ruleName": "FastConnection",
        "passedCount": 0,
        "failedCount": 0
      },
      {
        "ruleName": "NoobSegregation",
        "passedCount": 0,
        "failedCount": 0
      }
    ],
    "type": "MatchmakingCancelled",
    "message": "Cancelled by request.",
    "gameSessionInfo": {
      "players": [
        {
          "playerId": "player-1"
        }
      ]
    }
  }
}
```


MatchmakingFailed

对战票证遇到了错误。这可能是由于无法访问游戏会话队列或内部错误所致。

资源： ConfigurationArn

详细信息：类型、票证、ruleEvaluationMetrics、消息、matchId、gameSessionInfo

Example


```
{
  "version": "0",
  "id": "025b55a4-41ac-4cf4-89d1-f2b3c6fd8f9d",
  "detail-type": "GameLift Matchmaking Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2017-08-16T18:41:09.970Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws-cn:gamelift:us-west-2:123456789012:matchmakingconfiguration/
SampleConfiguration"
  ],
  "detail": {
    "tickets": [
      {
        "ticketId": "ticket-1",
        "startTime": "2017-08-16T18:41:02.631Z",
        "players": [
          {
            "playerId": "player-1",
            "team": "red"
          }
        ]
      }
    ],
    "customEventData": "foo",
    "type": "MatchmakingFailed",
    "reason": "UNEXPECTED_ERROR",
    "message": "An unexpected error was encountered during match placing.",
    "gameSessionInfo": {
      "players": [
        {
          "playerId": "player-1",
          "team": "red"
        }
      ]
    },
    "matchId": "3ea83c13-218b-43a3-936e-135cc570cba7"
  }
}
```

使用 FlexMatch 实现高安全性

AWS 的云安全性的优先级最高。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 客户，您也将从这些数据中心和网络架构受益。

安全性是 AWS 和您的共同责任。有关如何在使用 FlexMatch 时应用责任共担模式的信息，请参阅 中的 Amazon GameLift 安全性。

GameLift FlexMatch 发布说明和开发工具包版本

发布说明提供有关与服务相关的新增 GameLift 功能、更新和修复的详细信息。FlexMatch此页面还包括 GameLift 开发工具包版本历史记录。

GameLift 开发人员资源

要查看所有 GameLift 文档和开发人员资源，请参阅 文档Amazon GameLift主页。

AWS 词汇表

有关最新 AWS 术语，请参阅 AWS General Reference 中的 AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。