



# 動画アプリの投げ銭機能における 消耗型課金の仕組みと実装

Takeshi Ihara

2019/09/06 iOSDC 2019

お疲れ様です、「動画アプリの投げ銭機能における消耗型課金の仕組みと実装」というタイトルで発表させていただきます。  
稻見さんの圏論の話やLTの発表ではなく、地味なタイトルのセッションに来てくれてありがとうございます。

また、このトラックEの先ほどのロクネムさんの「詳細Auto-Renewable Subscriptions」から続けて見ている人は、同じ課金がテーマですが、Auto-Renewable SubscriptionsではなくConsumableつまり消耗型に関するトークなので、また少し違うものになると思います。  
続けて見る人はその差分も楽しんでみてください。

# Takeshi Ihara

---



- 2018.09.01 AbemaTV Join
- 2018.09.02 iOSDC 2018 LT  
「小さくはじめる端末管理」
- ボルダリングと格闘ゲームが趣味
- <https://twitter.com/nonchalant0303>

初めましての人は初めまして、AbemaTVでiOSエンジニアとして働いている井原岳志です。

ちょうど昨年の09.01にAbemaTVにジョインして、初の仕事が09.02の去年のiOSDCの「小さくはじめる端末管理」というLTでした。

退社したらボルダリングか格闘ゲームのどちらかを夜な夜なやっています。

先月に格闘ゲームの世界大会に出るためにラスベガスまで行ってきました。

たまにしか技術的なツイートはしないですが、趣味のツイートは多いので興味ある方はフォローしてくれると嬉しいです。

# サポート機能 (AbemaTV)



- 2019.04.22  
サポート機能を  
リリース！  
番組や番組出演者に  
スタンプを送って  
サポートができる！

今回のタイトルに含まれている動画アプリの投げ銭機能ですが、今年の4月にAbemaTVでサポート機能という名称でリリースしました。

AbemaTVで放送している番組や番組出演者にスタンプを送ってサポートできる機能です。

サポートされたスタンプは番組の制作費や出演者の出演料に上乗せされ、1ユーザーが番組や出演者に直接的に貢献することができます。

今年の1月から開発を開始して04.22リリースだったのと、04.14に開催された技術書展6にAbemaTVのiOSチームで出展したのと重なり、正直けっこうキツいスケジュールでした…

午前0:35 · 2019年4月21日 場所: 東京 世田谷区 · Twitter for iPhone

午後10:53 · 2019年4月23日 · Twitter for iPhone

午後6:48 · 2019年4月22日 · Twitter for iPhone

午前10:51 · 2019年4月23日 · Twitter Web Client

開発に関わったメンバーもリリースしたときに色々な思いが込もったツイートをしていました。

多くは語らないといった感じですね。

それ以降は会社の内外の人に色々と相談しつつプロジェクト全体でスケジュールや見積もりの方法を試行錯誤して、最近は平和な開発がでています。



サポートするぞ！

せっかくの機会ですので、実際にサポート機能を使ってみます。

デモの中はアプリ内の番組は再生されませんが、こちらはAbemaTVの機能で画面録画中は動画の再生はされないようにしています。

これはコンテンツの権利保護が目的の機能ですので、今回のデモには関係ないので気にしないでください。

ではさっそく見ていきましょう。

## Contents



1. App内課金
2. 課金形式による違い
3. AbemaTVにおける消耗型課金の設計
4. AbemaTVにおける消耗型課金の動作検証
5. まとめ

(04:20 ~ )

今回のトークですが、最初にApp内課金について軽く説明して、自動更新サブスクリプションや消耗型などの様々な課金形式の違いについて説明します。

その違いを受けて、AbemaTVにおける消耗型課金の設計の概要と意図を説明します。

また、課金処理なのでリリースした後の不具合は致命的なものになり得るので、リリース前にどのような動作検証したかを説明して、最後にまとめてこのトークは終了になります。

1

# App内課金



(04:50 ~ )

はじめにApp内課金について軽く説明します。

# App内課金とは

- App 内課金とは、iOS デバイスやコンピュータ上の App の中で追加のコンテンツや定期購読コンテンツを 購入できる仕組み

<https://support.apple.com/ja-jp/HT202023>

Appleの公式ドキュメントを参考にすると、App内課金とは「iOS デバイスやコンピュータ上の App の中で追加のコンテンツや定期購読コンテンツを 購入できる仕組み」です。

# 課金形式

- ・消耗型 (Consumable) - アプリ内通貨など  
一度使うとなくなり、再購入することができる
- ・非消耗型 (Non-Consumable) - 広告の非表示など  
一度の購入だけで無制限に使用できる
- ・自動更新サブスクリプション (Auto-Renewable Subscriptions) - 月額VODなど  
月間サービスや定期的にアップデートされたコンテンツを購入することができる
- ・非更新サブスクリプション (Non-Renewing Subscriptions) - シーズンパスなど  
期間限定のサービスやコンテンツを購入することができる

App内課金は消耗型、非消耗型、自動更新サブスクリプション、非更新サブスクリプションの4種類があります。

今回のテーマである消耗型はいわゆるソシャゲの魔法石などのアプリ内通貨などで用いられます。  
この課金アイテムは一度使うとなくなり、再購入することができます。

非消耗型は広告の非表示機能など、一度の購入だけで無制限に使用できます。

自動更新サブスクリプションは月額VODなどの定期的にアップデートされるコンテンツを購入するために用いられます。  
こちらを実装しているアプリや紹介している記事などが多いと思います。

非更新サブスクリプションは自動更新サブスクリプションに比べてあまり用いられていない印象なのですが、シーズンパスなどの期間限定のサービスに用いられます。  
このようにサービスが提供する性質により適切な課金形式を選ぶ必要があります。

# AbemaTVにおける自動更新サブスクリション



- ¥960/月でプレミアム機能を提供
- 1. すべての作品が見放題
- 2. 広告なしでビデオをすぐ再生
- 3. 放送中でも最初からみられる
- 4. ダウンロードで通信量をほぼゼロに
- 5. 放送後の番組でもコメントが楽しめる

AbemaTVでは2つの課金形式を用いてサービスを提供しています。

1つは自動更新サブスクリプションのプレミアム機能です。

こちらは月額960円でAbemaTVのすべての作品が見放題になり、ダウンロード機能など様々なプレミアムな機能を使用することができます。

# AbemaTVにおける消耗型課金



- ・アプリ内通貨(Abemaコイン)  
コインでスタンプを購入して、  
番組や番組出演者にスタンプを送って  
サポートができる！

もう1つは今回のテーマの消耗型課金です。

こちらは今年の04.22にサポート機能と同時に提供されました。

AbemaTVのサポート機能で用いるスタンプを購入するためのアプリ内通貨、Abemaコインを購入するために使用します。

今回はこちらの機能について設計などを中心に話していきます。

# 2

## 課金形式による違い



(06:50 ~ )

先ほどはそれぞれの課金形式の提供するサービスの性質などを説明しましたが、ここではシステム的な違いについて説明します。

# 課金形式による機能の違い

[https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Products.html#/apple\\_ref/doc/uid/TP40008267-CH2-SW2](https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Products.html#/apple_ref/doc/uid/TP40008267-CH2-SW2)  
[https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Restoring.html#/apple\\_ref/doc/uid/TP40008267-CH8-SW9](https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Restoring.html#/apple_ref/doc/uid/TP40008267-CH8-SW9)

	再購入の可否	レシートへの表示	デバイス間での同期	復元
消耗型	○ (何度でも)	再購入まで	同期なし	復元されない
非消耗型	✗	常に	システムにより同期	システムにより同期
自動更新サブスクリプション	✗ (自動的に更新される)	常に	システムにより同期	システムにより同期
非更新サブスクリプション	○ (更新する)	常に	Appにより同期	Appにより同期

この表はそれぞれの課金形式の「再購入の可否」「レシートへの表示」「デバイス間での同期」「復元」について記載しています。

—

システムにより同期 → レシートを更新するだけで購入情報を復元できる(SKReceiptRefreshRequest)

Appにより同期 → 完了したトランザクションをAppStoreに問い合わせる restoreCompletedTransactions)

# 課金形式による機能の違い

[https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Products.html#/apple\\_ref/doc/uid/TP40008267-CH2-SW2](https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Products.html#/apple_ref/doc/uid/TP40008267-CH2-SW2)  
[https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Restoring.html#/apple\\_ref/doc/uid/TP40008267-CH8-SW9](https://developer.apple.com/jp/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Restoring.html#/apple_ref/doc/uid/TP40008267-CH8-SW9)

	再購入の可否	レシートへの表示	デバイス間での同期	復元
消耗型	<input type="radio"/> (何度でも)	再購入まで	同期なし	復元されない
非消耗型	✗	常に	システムにより同期	システムにより同期
自動更新サブスクリプション	✗ (自動的に更新される)	常に	システムにより同期	システムにより同期
非更新サブスクリプション	<input type="radio"/> (更新する)	常に	Appにより同期	Appにより同期

ここで注目して欲しいのは消耗型の「再購入の可否」と「レシートへの表示」の項目です。

先ほど説明した通り消耗型は何度でも再購入できます。

アプリ内課金は購入した際にAppStoreからレシートが発行されて、そのレシートに購入履歴などが記されています。

その購入履歴がこの「レシートへの表示」という項目です。

消耗型のレシートへの表示は再購入までになっています。

これはどういう意味かというと

# 再購入が何度も出来る

- ・ローカルのレシートの購入履歴が上書きされる  
購入に応じたインセンティブをユーザーに付与できなくなる  
AppStoreによる課金処理は完了している  
→ 課金だけ発生してユーザーにインセンティブを  
付与していない状態

消耗型を再購入した際に、以前の購入履歴が上書きされるため、「レシートへの表示」が再購入までということになります。  
上書きされた購入履歴は復元することはできません。

他の課金形式は再購入ができないので、「レシートへの表示」が常にされるという表現になっています。

購入に応じたインセンティブ、今回の例でいうとAbemaコインですが、コインを付与する前にユーザーが再購入をしてしまうと、以前の購入履歴が上書きされるので、  
購入履歴を見てコインを付与することができなくなります。  
しかし、これはAppStoreによる課金処理自体は完了しているので、課金だけ発生してユーザーにインセンティブを付与していない状態になり、これはカスタマーサポートなどでクレームがくる緊急度が高い問い合わせになり得ます。

# レシート (初回購入時)

```
{  
  "receipt": {  
    ...  
    "in_app": [  
      {  
        "product_id": "com.nonchalant.consumable1",  
        "transaction_id": "100000551439498",  
        "purchase_date": "2019-09-01 00:00:00 Etc/GMT",  
        ...  
      }  
    ]  
  }  
}
```

[https://developer.apple.com/library/archive/releasenotes/General/ValidateAppStoreReceipt/Chapters/ReceiptFields.html#/apple\\_ref/doc/uid/TP40010573-CH106-SW1](https://developer.apple.com/library/archive/releasenotes/General/ValidateAppStoreReceipt/Chapters/ReceiptFields.html#/apple_ref/doc/uid/TP40010573-CH106-SW1)

参考までに実際のレシートを見てみましょう。

こちらは初回購入時に発行されるレシートです。

レシートはjson形式になっていて、in\_appというフィールドにはcom.nonchalant.consumable1というproduct\_idのアイテムの購入履歴が1つだけあります。

# レシート(再購入時)

```
{  
    "receipt": {  
        ...  
        "in_app": [  
            {  
                "product_id": "com.nonchalant.consumable1",  
                "transaction_id": "100000556349963",  
                "purchase_date": "2019-09-03 10:10:26 Etc/GMT",  
                ...  
            }  
        ]  
    }  
}
```

[https://developer.apple.com/library/archive/releasenotes/General/ValidateAppStoreReceipt/Chapters/ReceiptFields.html#/apple\\_ref/doc/uid/TP40010573-CH106-SW1](https://developer.apple.com/library/archive/releasenotes/General/ValidateAppStoreReceipt/Chapters/ReceiptFields.html#/apple_ref/doc/uid/TP40010573-CH106-SW1)

こちらは再購入した際のレシートです。

初回購入時と同様に、in\_appというフィールドにはcom.nonchalant.consumable1というproduct\_idのアイテムの購入履歴が1つだけあります。しかし、これはtransaction\_idとpurchase\_dateが異なる別の購入履歴になっています。

このように同じアイテムを再購入した際、以前の購入履歴は上書きされてしまいます。

# 再購入が出来る条件

- 同じ商品の課金トランザクションが終了している

課金トランザクション

→ SKPaymentTransaction

トランザクションが終了している

→ SKPaymentQueue.finishTransaction(\_:)



消耗型は再購入できると説明しましたが、ユーザーが再購入できるようになるためには条件があります。

それは同じ商品の課金トランザクションが終了しているということです。

こちらの画面は課金トランザクションが終了しておらず、再購入できないときに購入しようとすると表示されるダイアログになります。

それぞれの用語を具体的なクラスやメソッド名で説明すると、課金トランザクションはStoreKitにおけるSKPaymentTransactionを指しています。

トランザクションが終了しているということはStoreKitにおけるSKPaymentQueueのfinishTransactionが呼び出されているということです。

こちらは一度課金処理を実装したことがある人はご存知かと思います。

# 課金トランザクションの扱いの注意点

- ・インセンティブを付与するまでトランザクションを終了してはいけない

> Once you've delivered the product, your application must call finishTransaction: to complete the transaction. When you call finishTransaction:, the transaction is removed from the payment queue. To ensure that products are not lost, your application should deliver the product before calling finishTransaction:.

- ・購入履歴がレシートに常に表示される課金形式では

意識しなくても問題ない

非消耗型、自動更新サブスクリプション、非更新サブスクリプション

<https://www.rapid-ideas.com/previews/wsa/page9/files/StoreKitGuide.pdf>

AppleのIAP課金の実装ガイドを見てみると、インセンティブを付与するまでトランザクションを終了してはいけないと書いています。

「your application should deliver the product before calling finishTransaction:」の文章ですね。

本日はこの文章だけ覚えてもらったらほぼほぼ大丈夫です。

ネットなどの課金実装のブログなどを見てみると、インセンティブが付与する前のAppStoreにおける購入が終了したタイミングでトランザクションを終了しているものが多く、このような実装はガイドに書かれていることと反しています。

ただ、インセンティブを付与する前にトランザクションを終了しても問題ないケースがあります。

それは「レシートへの表示」が常にされる課金形式です。

すなわち、「非消耗型」「自動更新サブスクリプション」「非更新サブスクリプション」の3つになります。

これらはユーザーの動作で再購入できないので、どのタイミングでも購入履歴を正常に取得できます。

このような場合は実装が楽になるケースもあるので、ガイドに反しますがインセンティブを付与する前にトランザクションを終了するのもありかと思います。

しかし、消耗型でそのような実装をしてしまうと、サービスとして致命的な不具合を抱えてしまう可能性があるので注意が必要です。

# 3

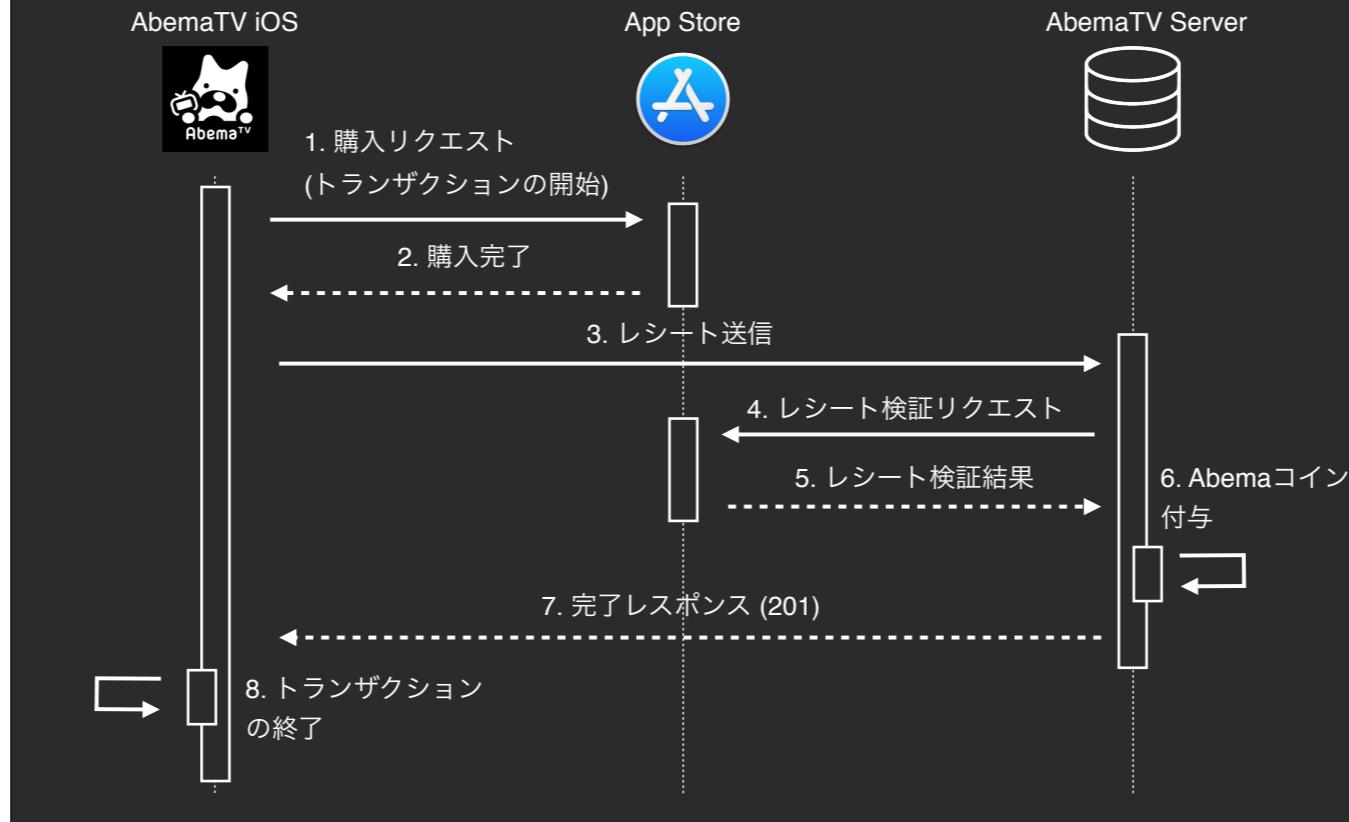
## AbemaTVにおける 消耗型課金の設計



(11:20 ~ )

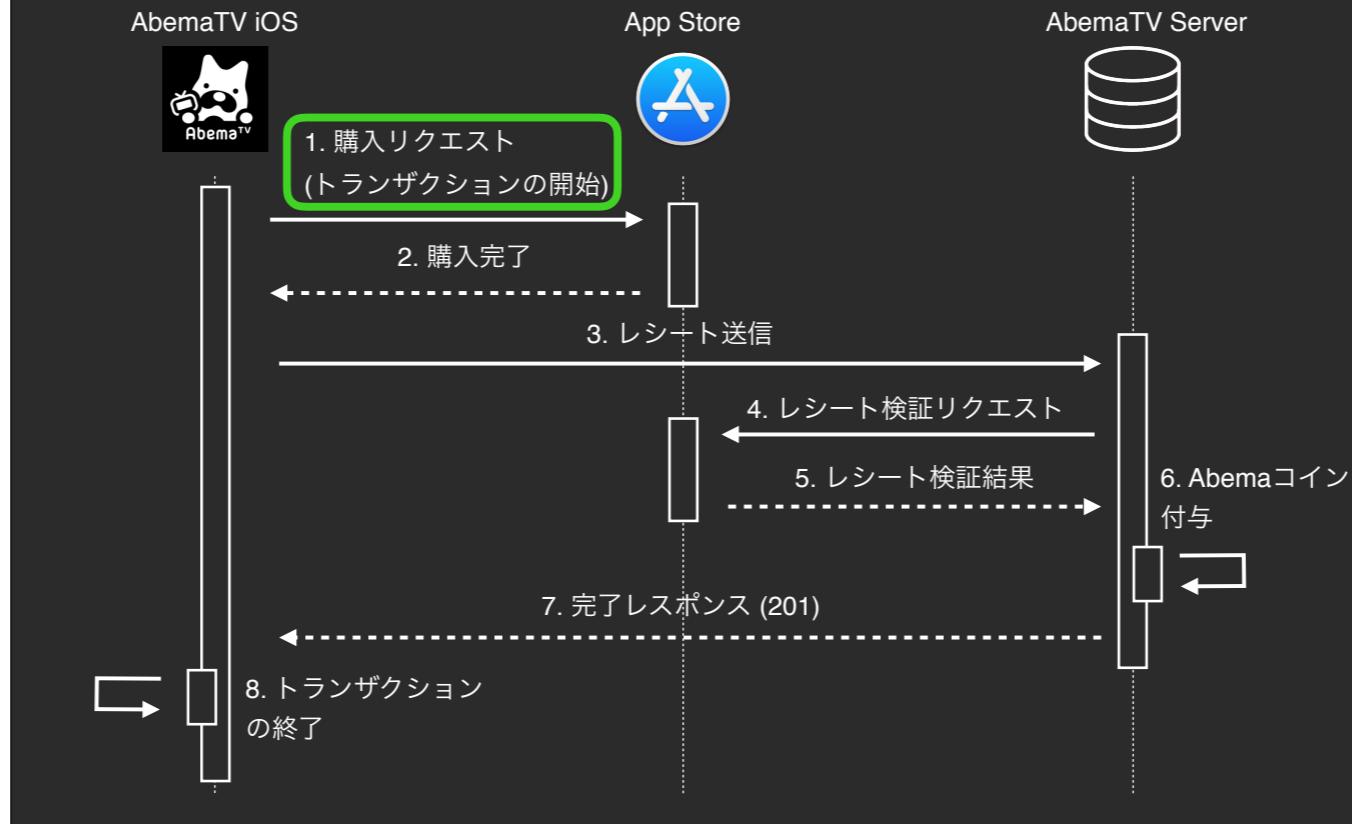
先ほどの課金トランザクションの扱いの注意点を踏まえて、AbemaTVにおける消耗型課金の設計を紹介します。

# 消耗型課金の購入フロー



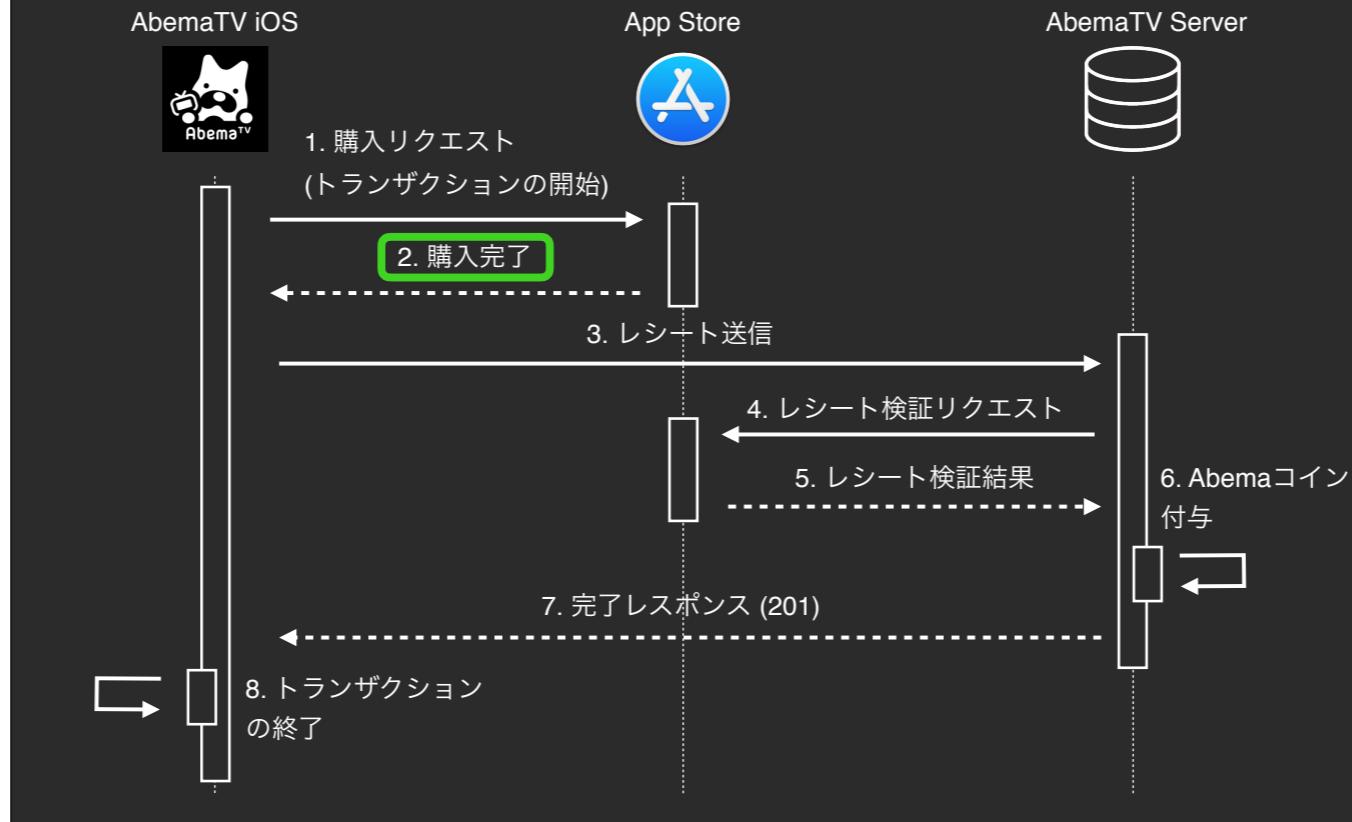
消耗型課金の購入フローは、クライアントの「購入リクエスト」すなわち「トランザクションの開始」から始まり「トランザクションの終了」までの8ステップで構成されています。

# 消耗型課金の購入フロー



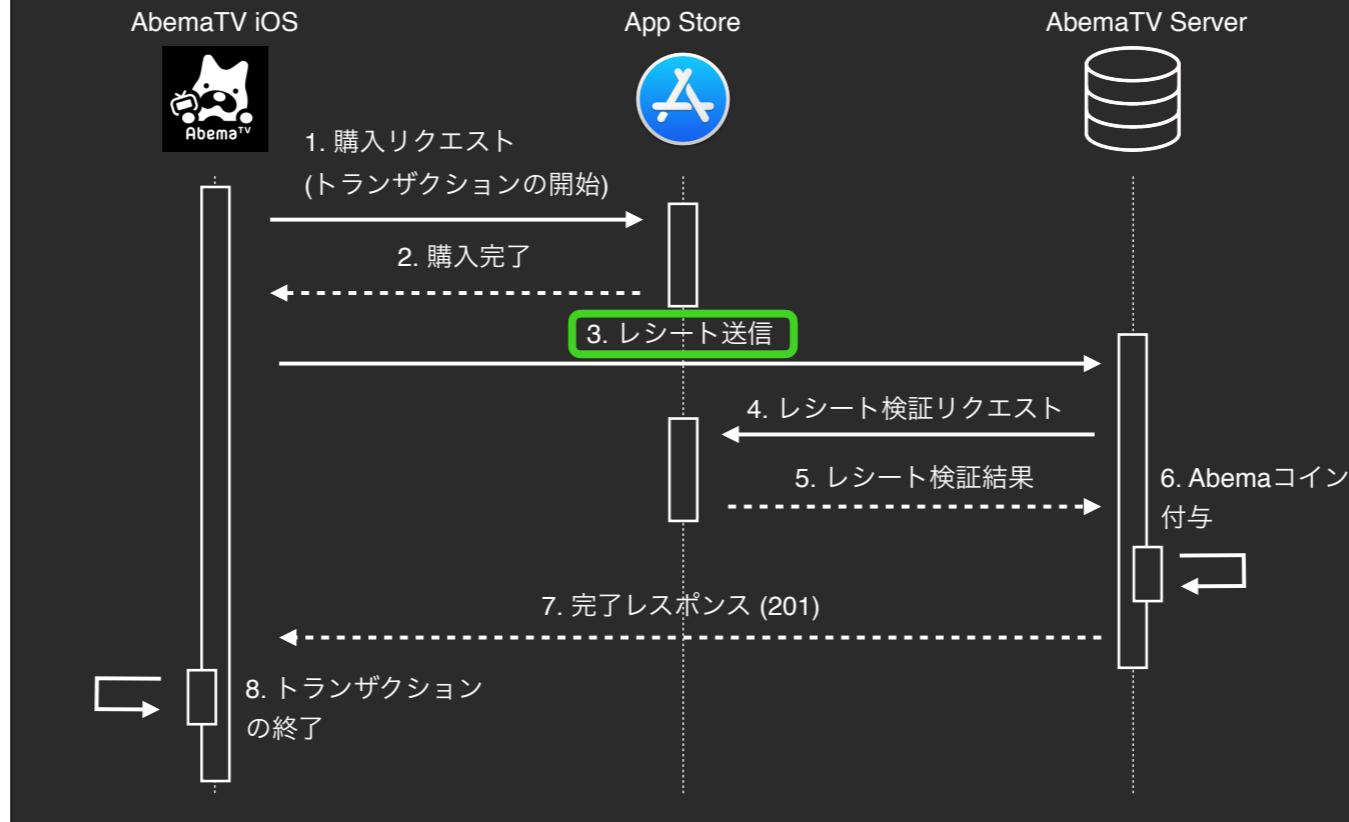
まず、ユーザーがAppStoreに対して消耗型の課金アイテムの購入リクエストをして、トランザクションが開始します。

# 消耗型課金の購入フロー



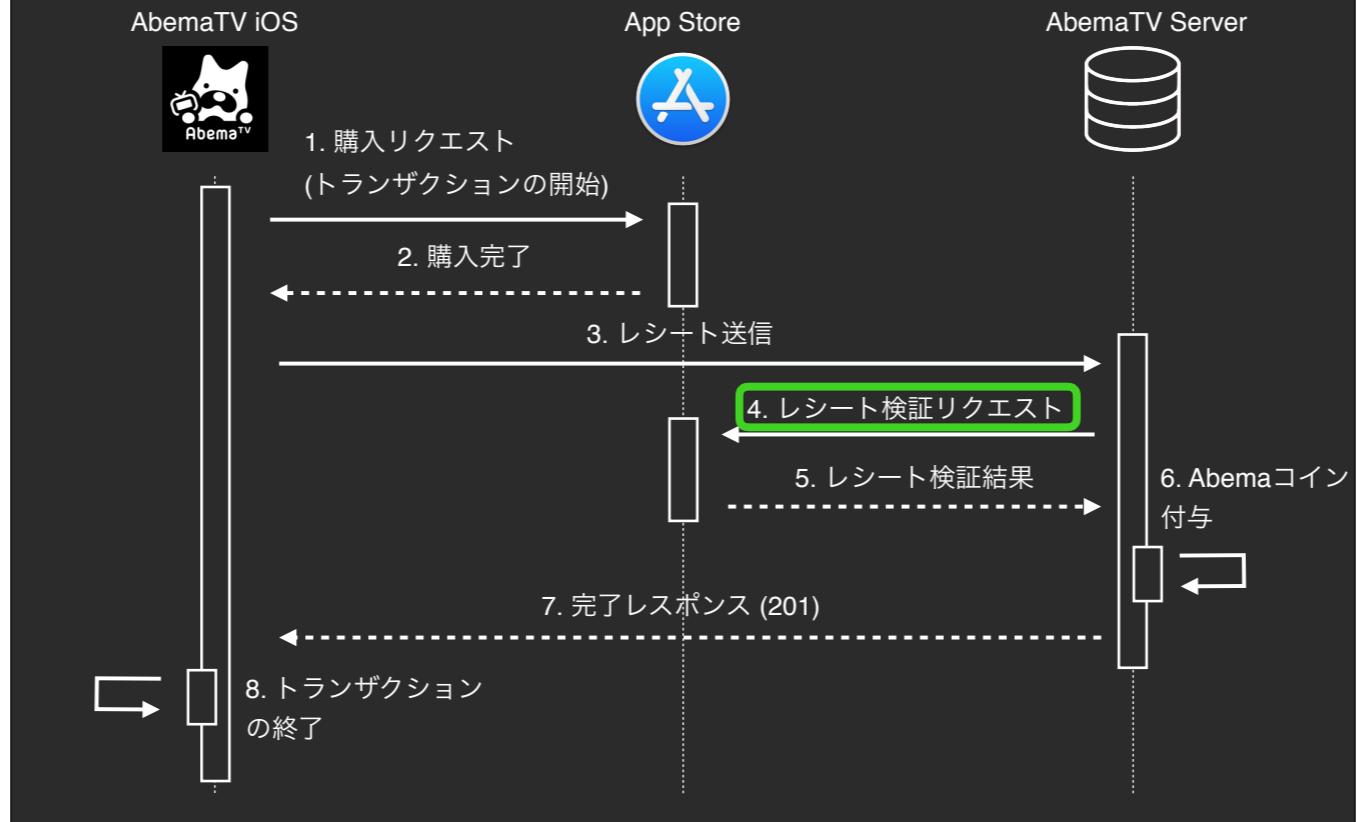
その後、ユーザーがFaceIDやFingerPrint、パスワード入力などの認証を行い、認証に成功すればAppStoreから購入完了のレスポンスが返ってきます。この時点でAppStoreによる課金は完了しています。

# 消耗型課金の購入フロー



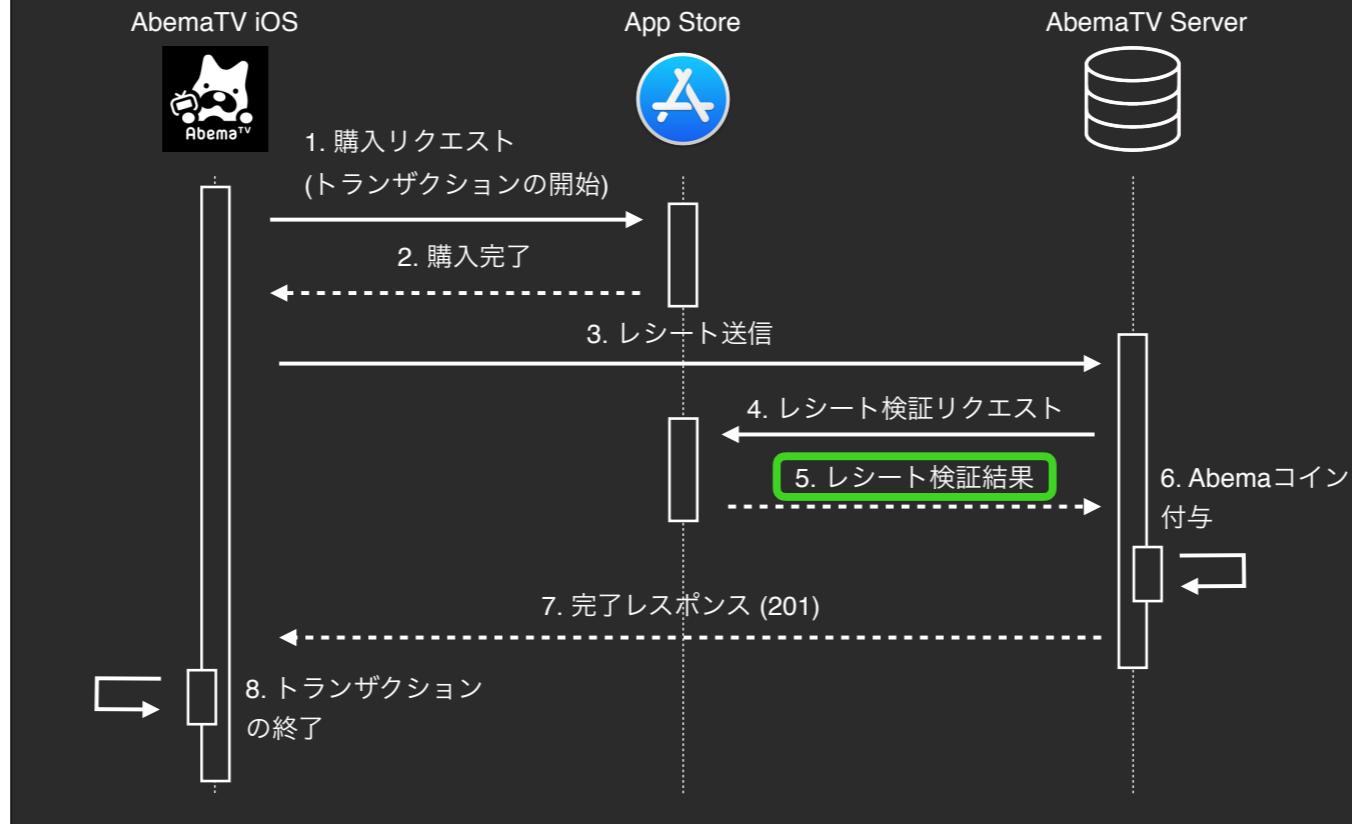
その後、トランザクションの終了をせずに、ローカルにData型で保存されているレシートをBase64でエンコードして弊社のサーバーに送信します。

# 消耗型課金の購入フロー



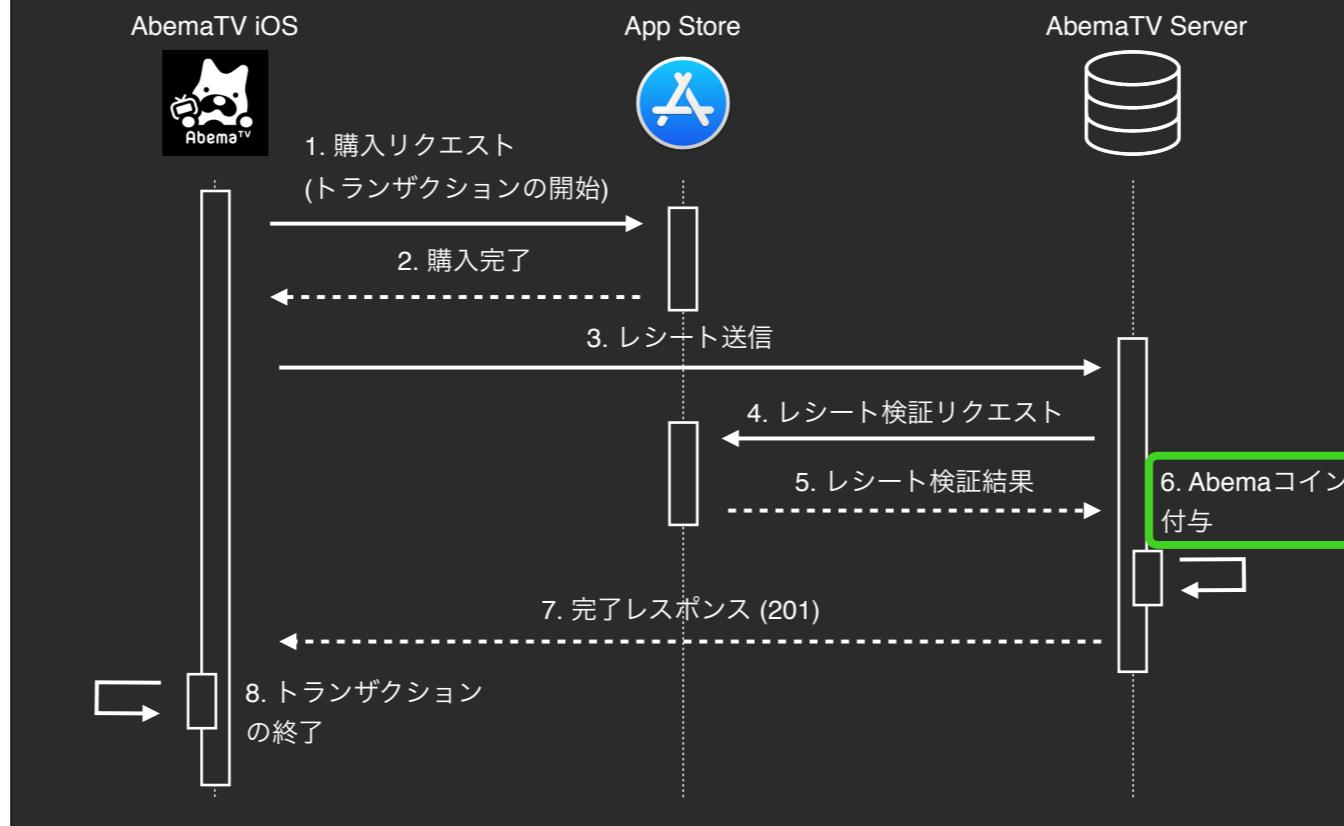
Base64でエンコードされたレシートを受け取ったサーバーは、そのレシートを用いてAppStoreに検証リクエストを投げます。

# 消耗型課金の購入フロー



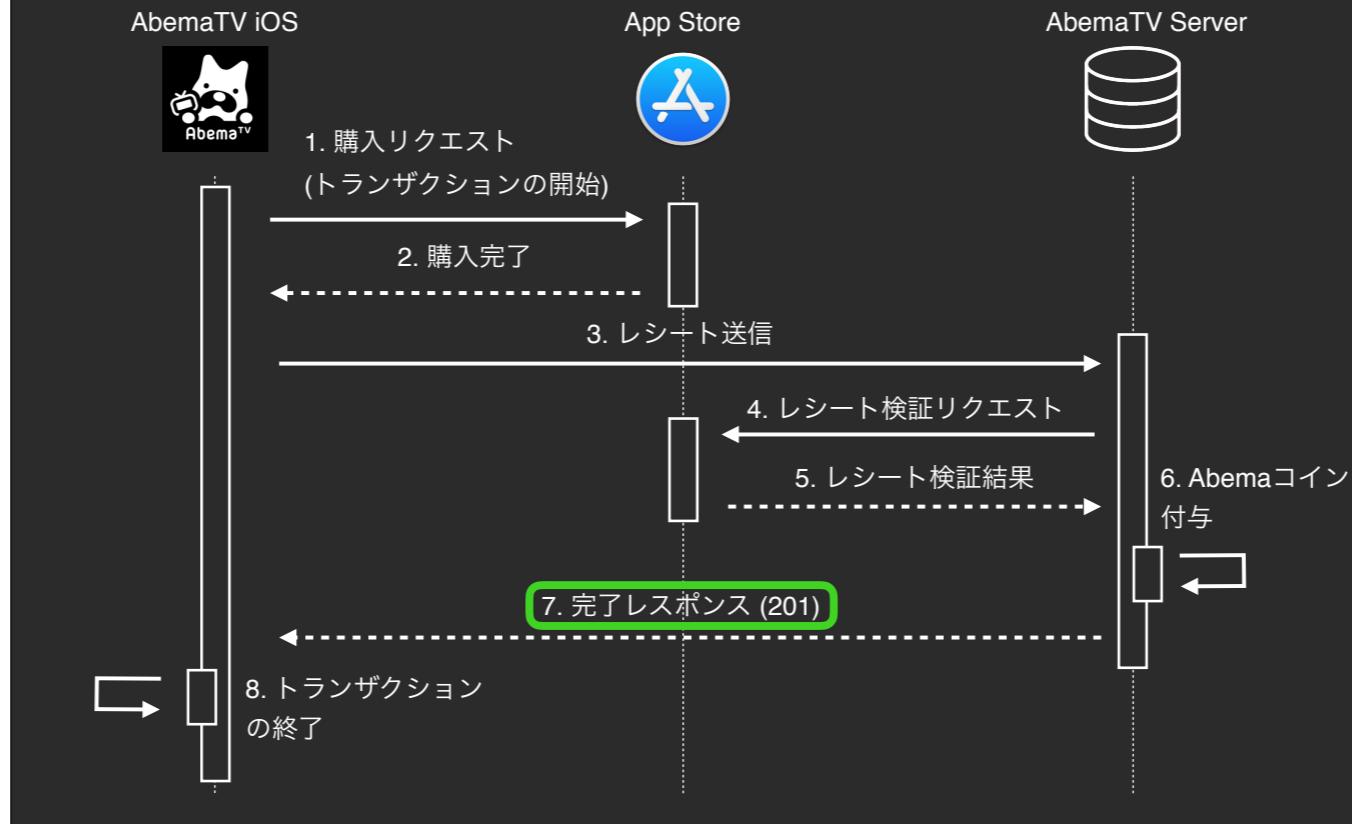
AppStoreは購入履歴を含んだレシート検証結果を弊社のサーバーに返却します。

# 消耗型課金の購入フロー



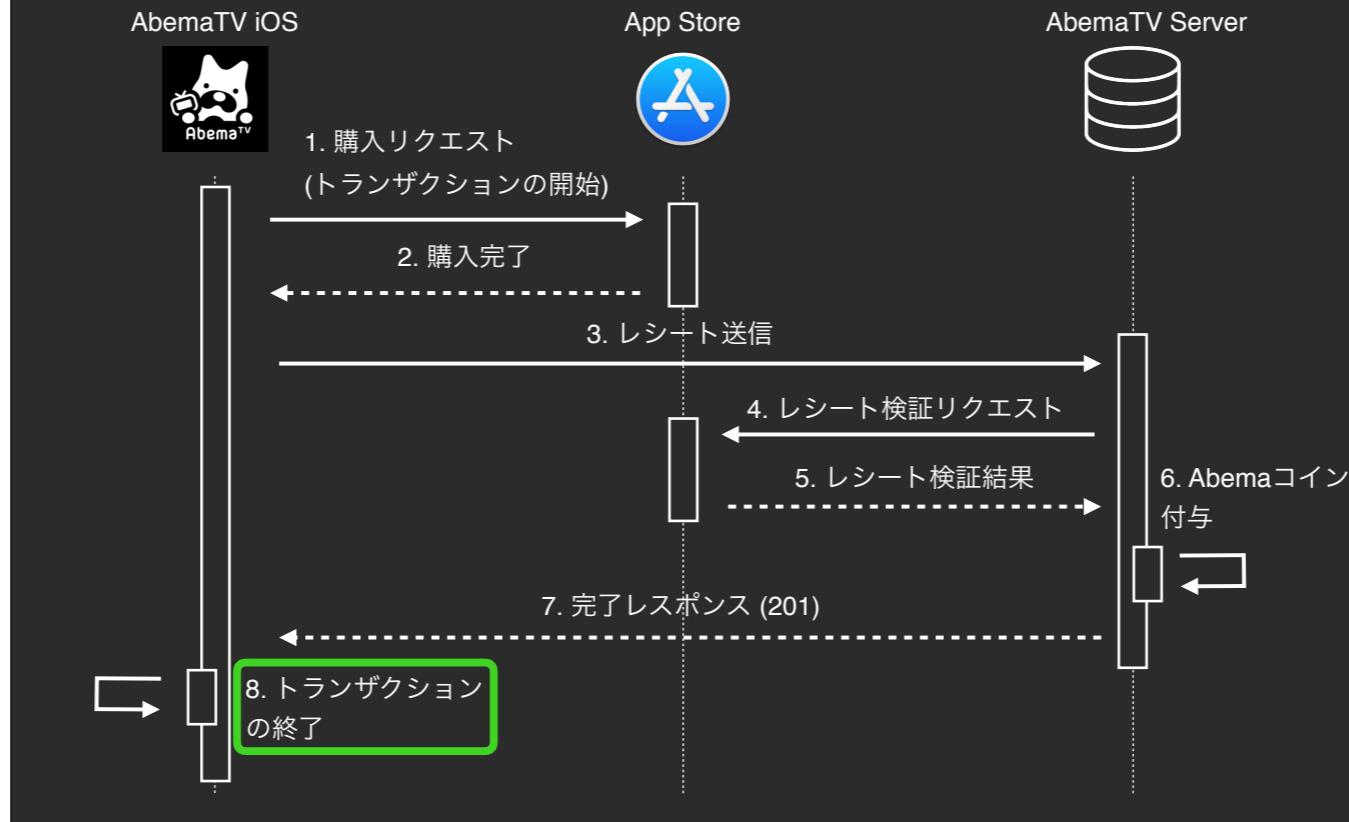
その検証結果に基づき、ユーザーにAbemaコインを付与します。

# 消耗型課金の購入フロー



ユーザーにAbemaコインの付与が完了したのち、完了レスポンスをクライアントに返します。

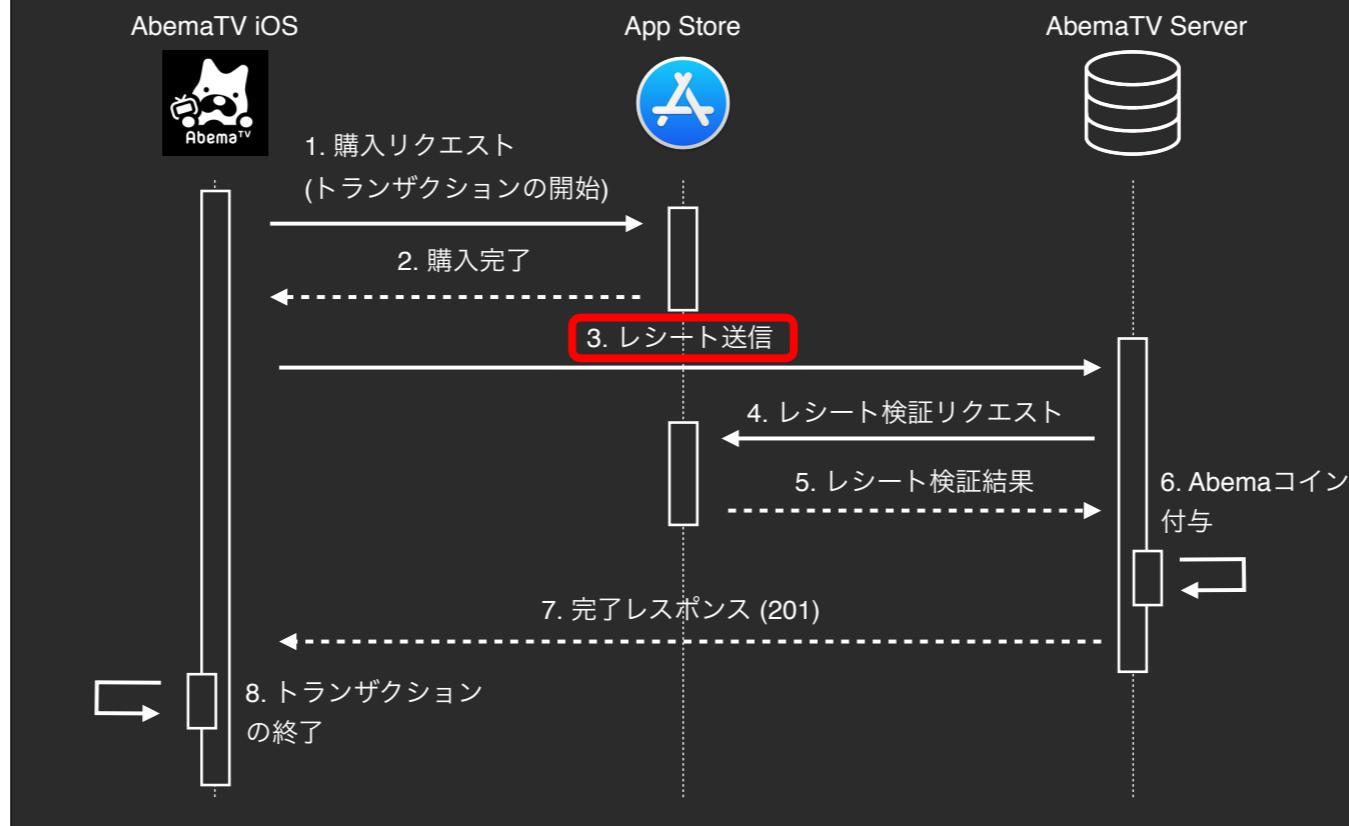
# 消耗型課金の購入フロー



そのレスポンスを受け取って、クライアントは課金トランザクションを終了します。

これが正常な消耗型課金の購入フローになります。

# 課金完了後に失敗するケース 1



しかし、このフローにはAppStoreの課金が完了した後に失敗するケースが2つ存在します。

1つめは3の弊社のサーバーにレシート送信をするステップになります。

# レシート送信に失敗するケース

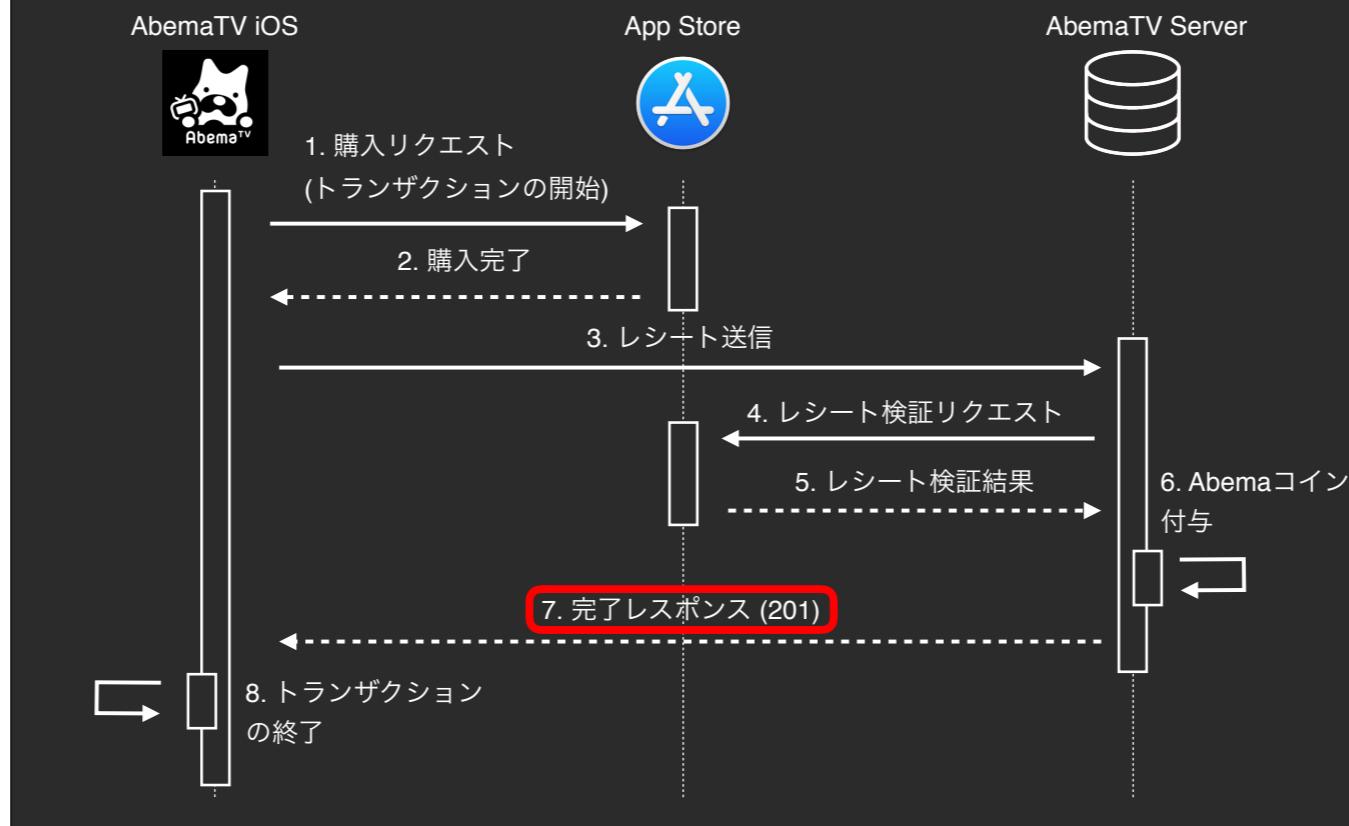
- ・ネットワークなどの問題でAPIリクエストの開始に失敗するケース
- ・AppStoreによる課金は発生している
  - ・サーバー側でAbemaコイン付与が出来ていない
  - ・課金トランザクションは終了していない  
ユーザーが再購入できない

レシート送信に失敗するケースは、クライアントのネットワーク環境など様々な原因でAPIリクエストの開始に失敗するケースです。

こちらは前の1と2のステップでAppStoreによる課金は完了しているにも関わらず、Abemaコインは付与できません。  
つまりはユーザーからお金だけいただいているという状態になります。

しかも、課金トランザクションは終了していないので、ユーザーは同じ商品を再購入することもできません。  
課金が一切できなくなり詰みの状態になってしまいます。

# 課金完了後に失敗するケース 2



2つめの失敗するケースは7の弊社のサーバーからのレスポンスを受け取るステップになります。

# 完了レスポンスに失敗するケース

- ・アプリのクラッシュなどで  
APIレスポンスの受信に失敗するケース
- ・AppStoreによる課金は発生している  
・サーバー側でAbemaコイン付与が完了している可能性がある  
・課金トランザクションは終了していない  
　　ユーザーが再購入できない

完了レスポンスの受け取りに失敗するケースは、APIリクエストは開始していますが、レスポンスを受け取る前にアプリがクラッシュしたり、サーバーがタイムアウトしたりなどのケースです。

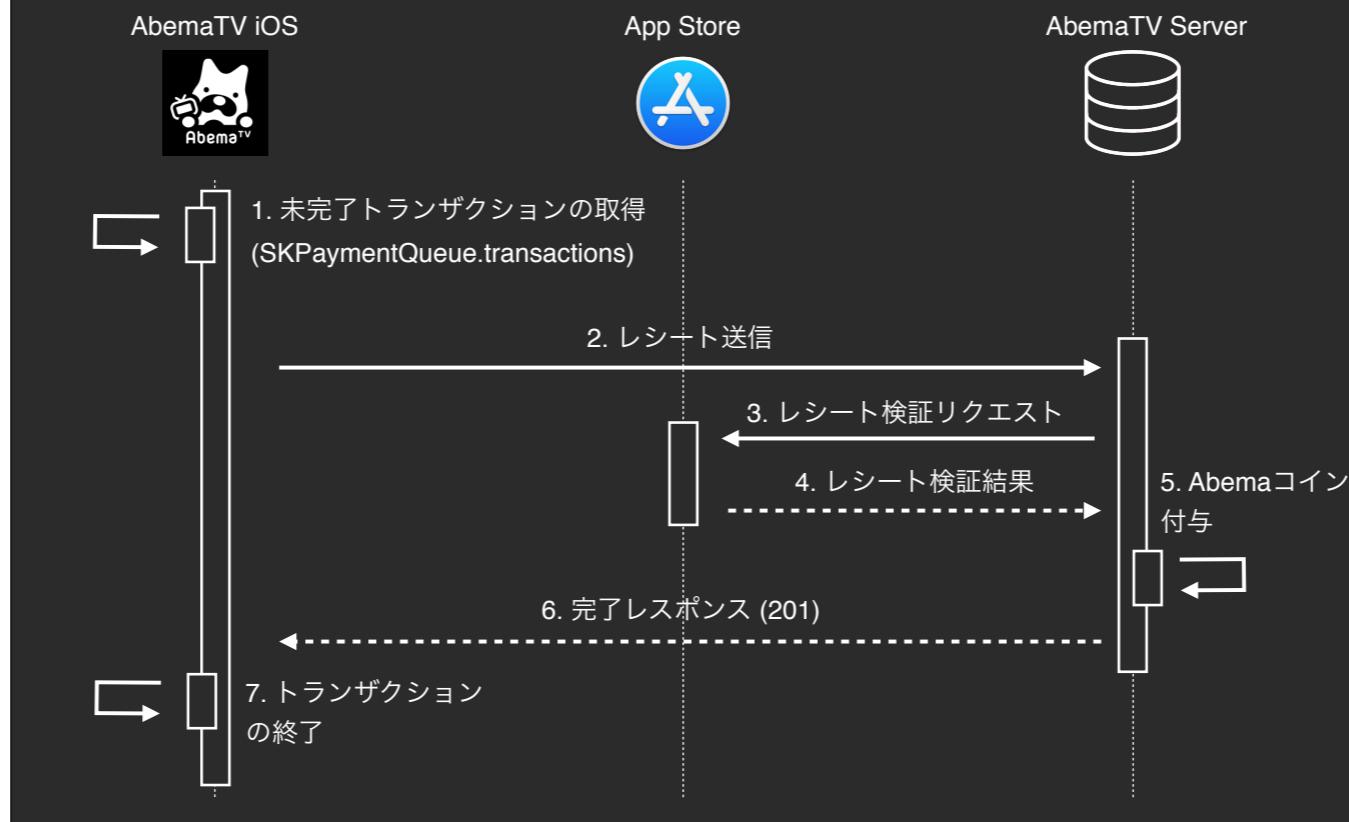
こちらも同様に前の1と2のステップでAppStoreによる課金は完了しているにも関わらず、Abemaコインを付与できていない可能性があります。リクエストは開始しているので、サーバー側でAbemaコインを付与できている可能性はありますが、レスポンスは返ってきていません。これもユーザーからお金だけいただいている可能性があるという状態になります。

さらに、課金トランザクションは終了していないので、ユーザーは同じ商品を再購入することもできません。  
こちらも課金が一切できなくなり詰みの状態になってしまいます。

# 未完了トランザクションの リトライが必要

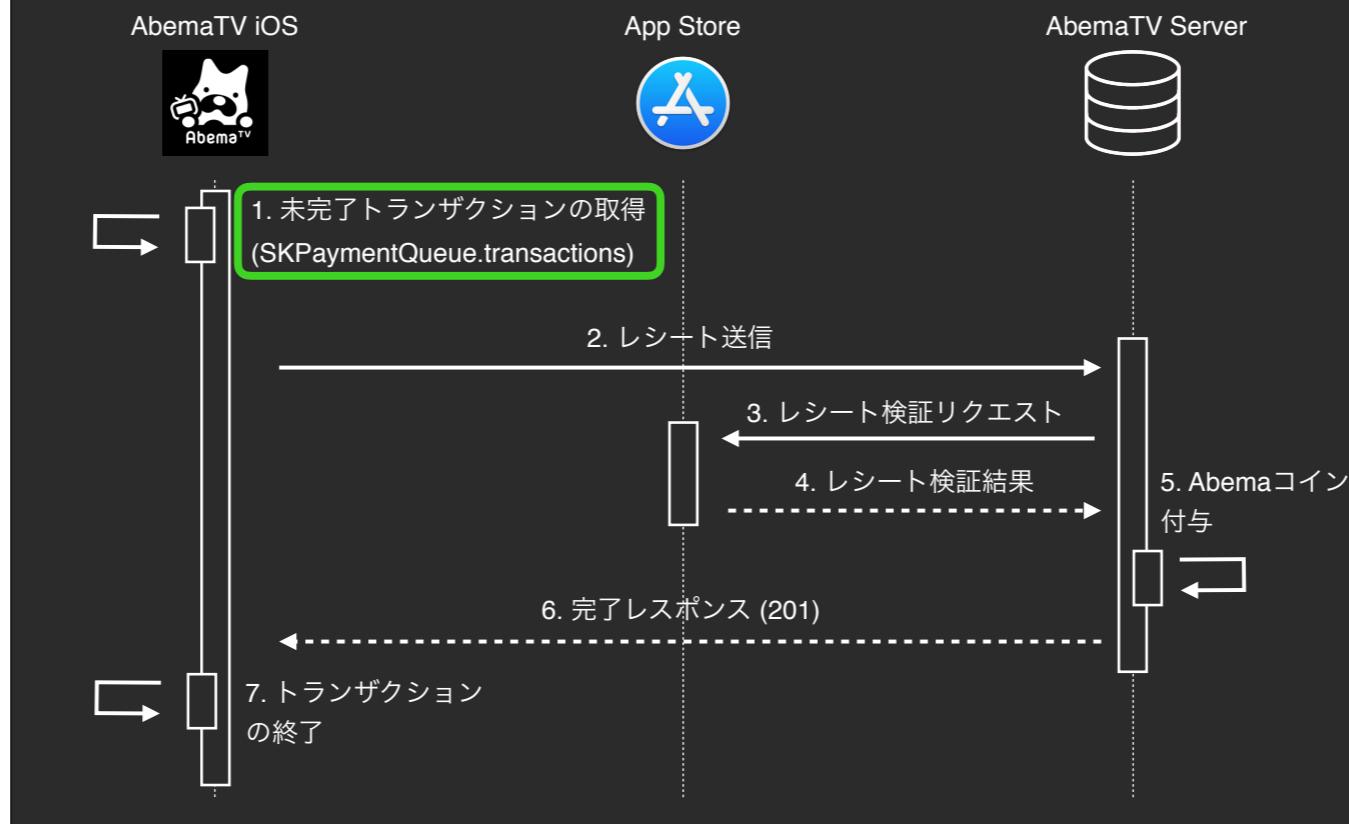
これらの2つのケースが発生した際の解決手段として、トランザクションの終了が呼ばれていない未完了トランザクションのリトライ機構が必要になります。

# Abemaコイン未付与のリトライ



未完了のトランザクションが存在するケースも2種類に分類でき、Abemaコインが未付与か付与済みの2つのケースがあります。こちらはAbemaコインが未付与のケースです。

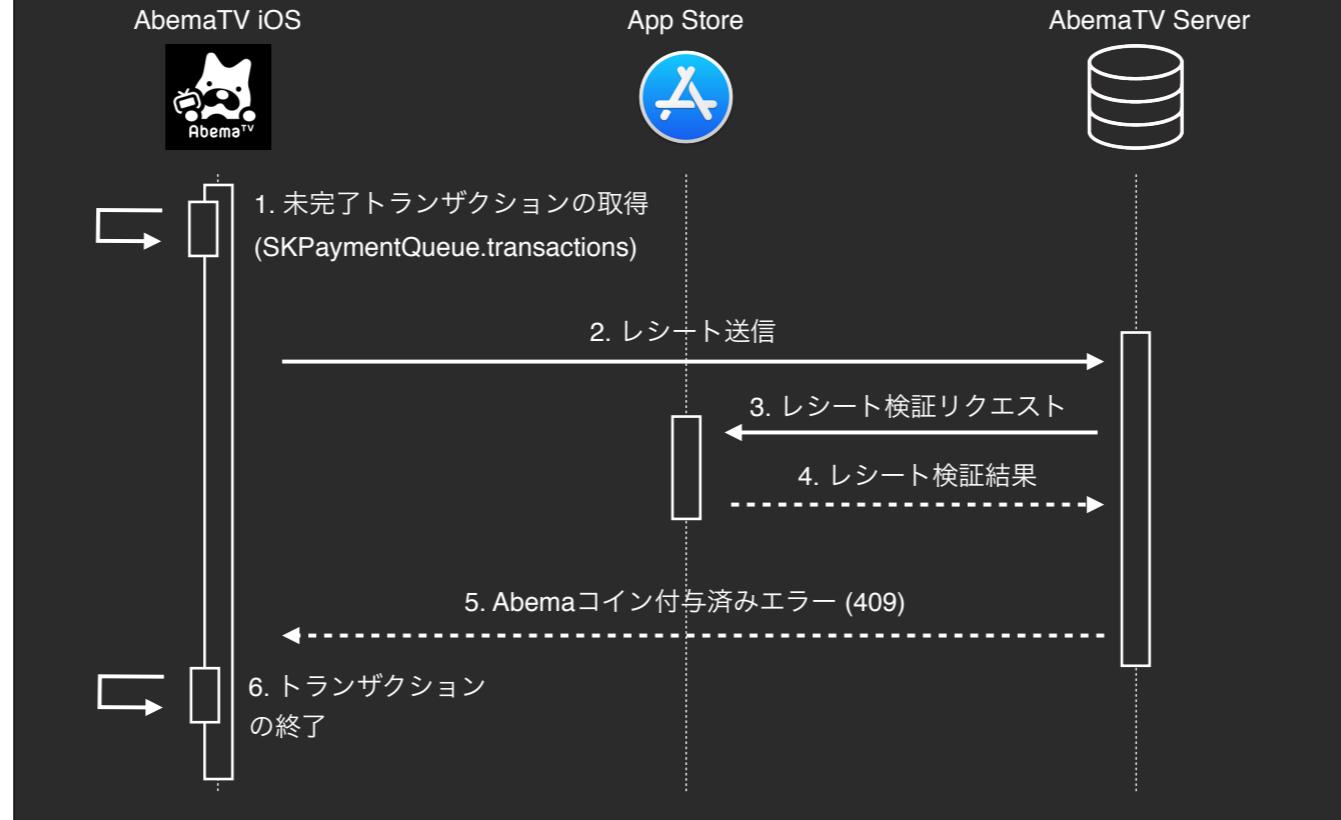
# Abemaコイン未付与のリトライ



まず、クライアントは`SKPaymentQueue.transactions`から未完了トランザクションを取得し、レシート送信をします。その後は正常の購入フローと同様に、サーバーからのレスポンスを受け取ったらトランザクションを終了します。

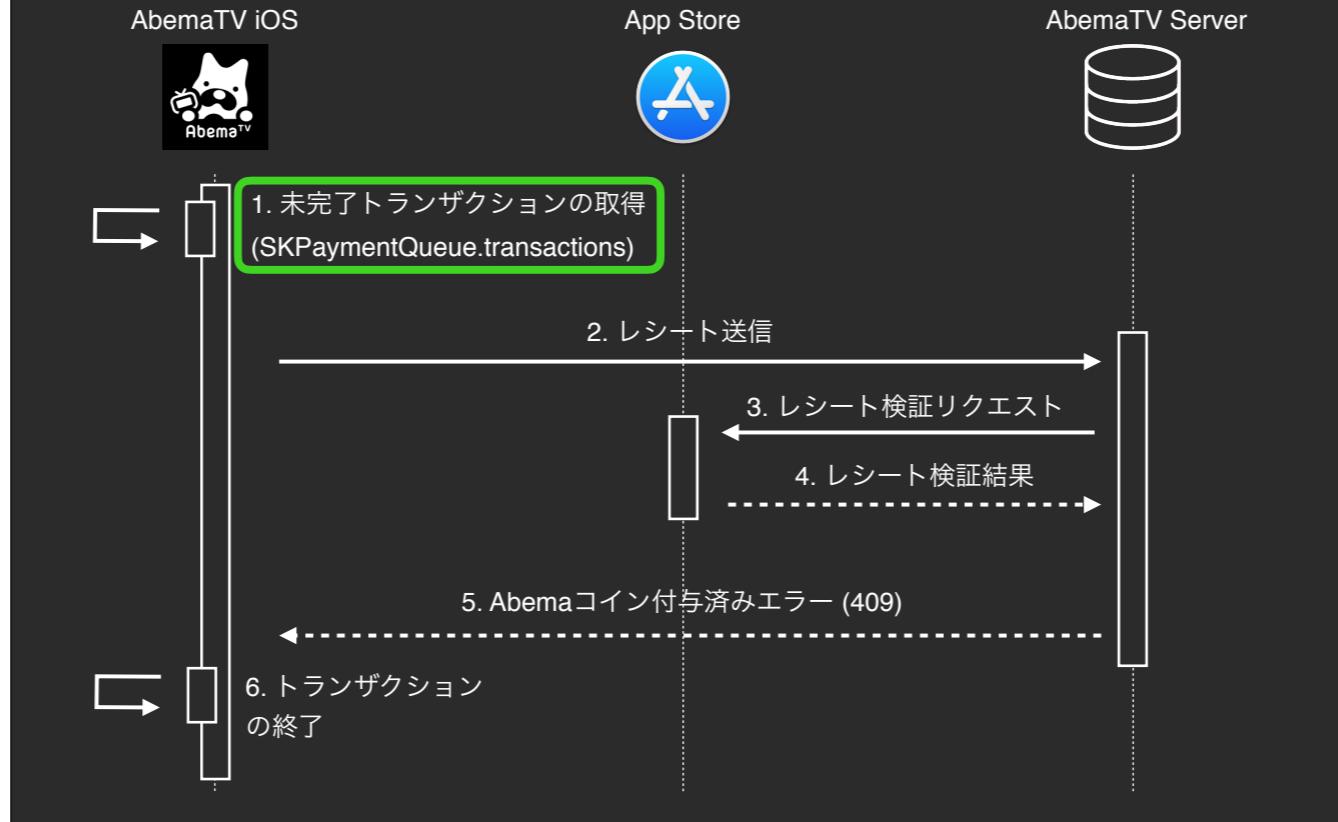
これで、Abemaコイン未付与の未完了トランザクションを正常に終了させることができます。

# Abemaコイン付与済みのリトライ



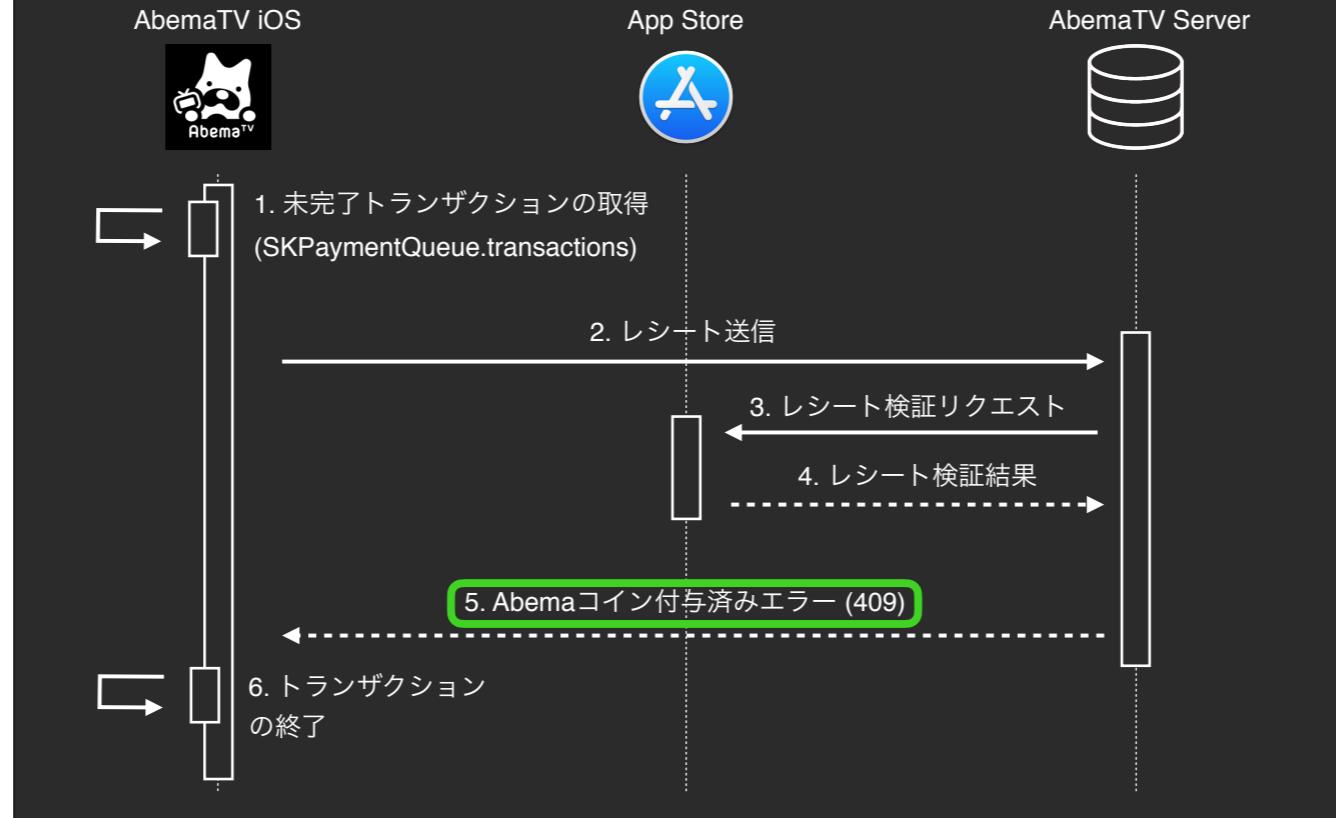
次はAbemaコインはすでに付与されていますが、サーバーからのレスポンスを受け取れず、トランザクションが未完了になっているケースです。

# Abemaコイン付与済みのリトライ



こちらは先ほどと同様に未完了のトランザクションを取得し、レシートを送信します。

# Abemaコイン付与済みのリトライ



サーバーはレシートの検証後にその購入履歴に対してAbemaコインはすでに付与しているので、Abemaコイン付与済みを意味するステータスコード 409を返却します。

クライアントはこのステータスコードを見て、トランザクションを終了します。

これで、Abemaコイン付与済みの未完了トランザクションも正常に終了させることができます。

トランザクションが  
終了している  
→ Abemaコインが  
付与されている

このようにして未完了トランザクションのリトライ処理を行うことにより、トランザクションが終了しているならばAbemaコインが付与されているという状態を保証します。

# リトライ処理の呼ぶタイミング

- 起動時、バックフォア時、コイン残高表示時に  
リトライ処理を行う

ユーザーがコイン残高を確認する前にリトライするようにして、  
不整合に気付かせづらくさせている

このリトライ処理の呼ぶタイミングですが、起動時、バックフォア時、コイン残高表示時に行っていきます。

購入時の即時リトライだとネットワークやサーバー不調などが即時に解決されないことが考えられるため、起動時やバックフォア時やコイン残高表示時などユーザーが自分のコイン残高を確認する直前にリトライするようにして、ユーザーに不整合を気付かせづらくさせています。

ユーザーから課金だけされたなどのお問い合わせがきた際は、再起動などのリトライ処理が行われる動作をお願いしています。

# リトライ処理の注意点

- 未完了トランザクションはSKPaymentQueue.transactionsで取得できる

購入中のトランザクションも含まれる

- 正常な購入フローの際はリトライ処理を行わないなどの処理が必要



先ほど未完了トランザクションはSKPaymentQueue.transactionsで取得できると説明しましたが、この未完了トランザクションには現在購入中のトランザクションも含まれます。

こちらは失敗談なのですが、SKPaymentQueue.transactionsに購入中のトランザクションも含まれることを気づかず、リトライ処理を最初に実装しました。AbemaTVのコイン購入画面はコイン残高が表示されているので、リトライ処理が行われます。つまりは購入中のトランザクションのリトライも行ってしまい、全てのトランザクションのレシート送信を最低でも購入時と購入時のリトライの2回行うという実装になっていました。

レシート送信を2回行うので、どちらかでAbemaCoinが付与されて、もう片方は付与済みのエラーが返ってきていました。

こちらはサーバー側のログで明らかにAbemaCoin付与済みのステータスコード 409が起きすぎているということで発覚しました。

動作的には致命的でないのですが、無駄にサーバーに負荷をかけたりログを汚してしまうので、正常な購入フローの際はリトライ処理を行わないように実装しました。リトライ処理を実装する際は注意が必要です。

# 4

## AbemaTVにおける 消耗型課金の 動作検証



(18:10 ~ )

消耗型課金の設計をざっくりと説明したので、こちらを見返せば消耗型課金を問題なく実装できるようになっていると思います。

しかし、先ほどの設計が正しく実装されていることをリリースする前に検証する必要があります。

課金周りの処理なので、致命的な実装ミスがあるとSNSやカスタマーサポートが大変になってしまいます。

ここからは実際にどう検証を進めたのかというお話がメインになってきますので、気軽に聞いてください。

# 検証項目の認識合わせ

- ・ 購入フローがトランザクションなどの内部の状態に依存しているので、QAチームにシーケンス図を渡しても検証項目を作成するのが難しい
- ・ 実際にどのような状況でどのような操作をすれば、どのような状態になるかの認識合わせをしないと検証項目を作成できない

消耗型課金の実装は無事に完了したのですが、QAチームとの認識合わせが大変でした。

なぜなら、購入フローがトランザクションなどの内部の状態に依存しているので、QAチームにシーケンス図を渡しても具体的にどうなるかイメージがつきづらいかったためです。

課金処理ということもあり慎重に進めたかったので、QAチームとサーバー担当者と自分の3プラットフォーム間で認識合わせをしつつ検証項目を作成することにしました。

そこで意識したのは、QAチームが実際にどのような状況でどのような操作をすればどのような状態になるかを言語化することでした。

# Mind Meister

- ・マインドマップ作成ツールのMind Meisterを用いて  
認識合わせをしました

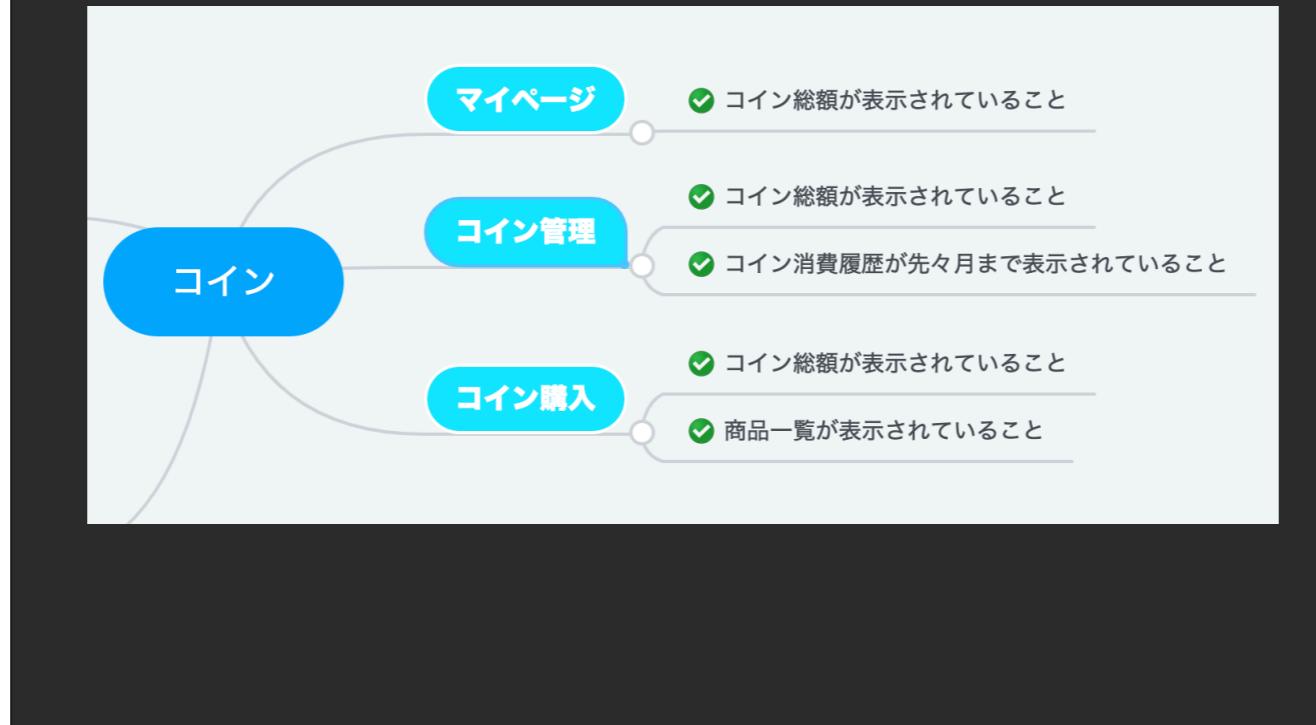


認識合わせのためにマインドマップ作成ツールのMind Meisterを用いました。

こちらは実際に用いたものを模したものなのですが、特にログインなどは必要ないので、興味がある方はこちらのQRコードを読み取って見てください。

では、実際にどのように認識合わせをしたか見ていきましょう。

# コイン周りの表示系

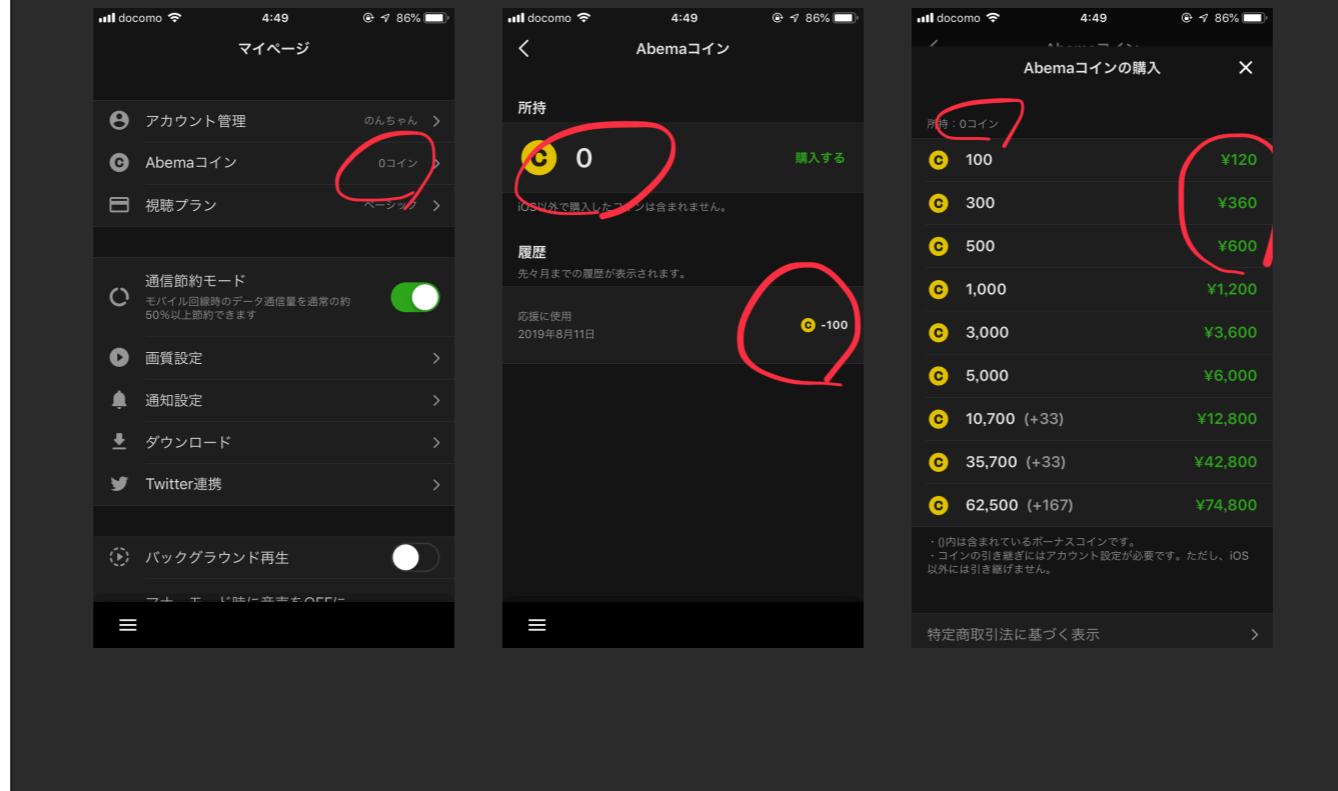


こちらはシンプルな表示系の検証項目になります。

この緑のチェックマークが付いている項目が検証項目になります。

マイページとコイン管理とコイン購入の3画面でそれぞれコイン周りの表示が正常に表示されていることの検証項目になります。

# コイン周りの表示系



実際の画面はこちらになります。

左からマイページ、コイン管理、コイン購入画面になります。

赤い丸で囲われている箇所が検証する箇所になります。

# 成功する購入フロー



次はコイン購入画面の成功する購入フローの検証項目になります。

動作としては商品を選択して購入してコイン残高が更新されていることの確認になります。

こちらの星マークが付いている項目が、実際の操作になります。

このようにQAチームが通常業務で行うような前提条件と入力と出力を明確化するようことを意識しました。

こちらは一切トランザクションやレシートという用語は用いていないのですが、購入フローのシーケンス図も一緒に渡していたので、最終的にQAチームも内部のシステムも理解していました。

実装側はシステムから設計・実装をして、デバッグやテストで確認するのですが、QAチームは逆に検証動作からシステムを理解するというフローのほうが理解しやすそうでした。

# 失敗する購入フロー



こちらは失敗する購入フローなのですが、項目が多いので実際にMind Meister側で見ていきましょう。

<https://www.mindmeister.com/1309256136?t=X6wMB2GxG6>

# 購入フローの動作検証

---

- 消耗型課金の購入フローを実装したサンプルプロジェクト

<https://github.com/Nonchalant/Consumable>

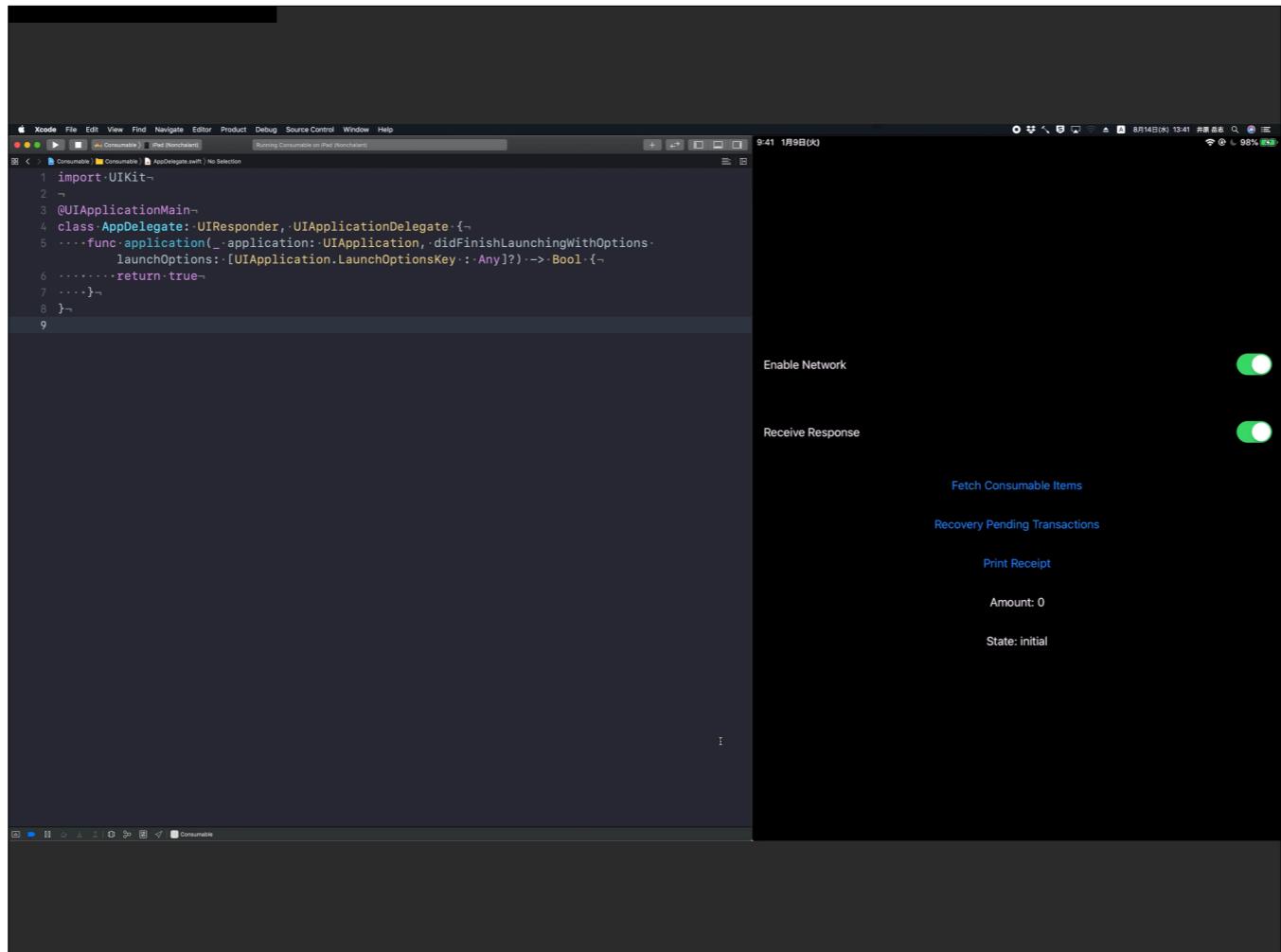
- Consumable In-App Purchase
- Combine
- SwiftUI

では実際にサンプルプロジェクトを用いて購入フローの動作検証をしてみます。

最初はこのサンプルプロジェクトはプロダクションの方からコピーして楽に作ろうと思っていたのですが、途中で飽きてしまったので、CombineとSwiftUIを用いて実装してみました。

はたしてこれはサンプルなのかという結果になってしまいました。

では実際に見ていきましょう。



(動画を流しつつ説明する)

このデモはあくまで一例なので、実際にはもっと多様なパターンでQAしています。

5

まとめ



Abema<sup>TV</sup>

(29:50 ~ )

最後にまとめです。

# まとめ

---

- ・消耗型課金はインセンティブを付与するまでトランザクションを終了しない  
再購入すると購入履歴が上書きされるため
- ・購入フローで失敗があるのでリトライ機構が必要  
正常な購入導線上でリトライしない

いろいろと話しましたが、この発表の大事なポイントは2つです。

1つめは「消耗型課金はインセンティブを付与するまでトランザクションを終了しない」ということ。  
これは再購入すると購入履歴が上書きされてインセンティブが付与できなくなるためです。

2つめは購入フローで失敗があるのでリトライ機構は必要です。  
ただ、リトライ機構を正常な購入導線上で動作しないように気をつけましょう。



以上になります、ご静聴ありがとうございました。