SBER TECH

One Nio vs кастомная Java-сериализация

Андрей Чернов, Java архитектор в СберТехе
@ chernovaf@mail.ru
chernovaf

О себе

Андрей Чернов

18 лет опыта разработки
Начинал на C/C++

Кандидатская диссертация
по индексам в БД

10 лет в СберТехе
Сейчас Java-архитектор

SBER TECH

План

1. **Что за One Nio и почему это круто**
2. Проблема с кастомными методами Java-сериализации
3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)
4. И что со скоростью? Разгон сериализации

План

1. *Что за One Nio и почему это круто*
2. **Проблема с кастомными методами Java-сериализации**
3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)
4. И что со скоростью? Разгон сериализации

План

1. *Что за One Nio и почему это круто*
2. *Проблема с кастомными методами Java-сериализации*
3. **«Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)**
4. И что со скоростью? Разгон сериализации

План

1. *Что за One Nio и почему это круто*
2. *Проблема с кастомными методами Java-сериализации*
3. **«Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)**
   - 3.1. Как реализовали defaultWriteObject() и defaultReadObject()?
   - 3.2. Как обеспечили десериализацию зацикленных структур данных?
   - 3.3. Как обошли ограничения JPMS?
4. И что со скоростью? Разгон сериализации

SBER TECH

План

1. *Что за One Nio и почему это круто*
2. *Проблема с кастомными методами Java-сериализации*
3. *«Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)*
4. **И что со скоростью? Разгон сериализации**

План

1. **Что за One Nio и почему это круто**
2. Проблема с кастомными методами Java-сериализации
3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)
4. И что со скоростью? Разгон сериализации

Что за библиотека One Nio?

- Highload обмен данными между серверами.
- Java-сериализация – одна из основных частей.

GitHub

https://github.com/odnoklassniki/one-nio

Что за библиотека One Nio?

- Highload обмен данными между серверами.
- Java-сериализация – одна из основных частей.

GitHub

https://github.com/odnoklassniki/one-nio

- Форкнули и развиваем в СберТехе.
- Используем в своём сервисе Platform V SessionsData.

Чем крута библиотека One Nio?

Очень быстрая

Быстрее других библиотек Java-сериализации.

Используются классы:
- MagicAccessorImpl
- Unsafe

Очень гибкая

Не требует схемы данных в отличие от:
- Google Protocol Buffers
- Apache Avro
- Apache Thrift
- etc.

Умеет сериализовать любой Java Object.

Чем крута библиотека One Nio?

Java-сериализация: максимум скорости без жёсткой структуры данных

Как мы упростили жизнь высоконагруженным сервисам. Часть 1

Основная фишка библиотеки One Nio

«Из коробки» ~ 50 отличных сериализаторов для стандартных классов:

primitives, String, arrays, Collections, Maps, etc.

Скрываются за абстракцией:

```
public abstract class Serializer<T> implements Externalizable {
    public abstract void calcSize(T obj, CalcSizeStream css) throws IOException;
    public abstract void write(T obj, DataStream out) throws IOException;
    public abstract T read(DataStream in) throws IOException, ClassNotFoundException;
    public abstract void skip(DataStream in) throws IOException, ClassNotFoundException;
    public abstract void toJson(T obj, StringBuilder builder) throws IOException;
    ...
```

Основная фишка библиотеки One Nio

Для каждого нестандартного класса генерируется его личный сериализатор:

```
public class GeneratedSerializer extends Serializer<Object> {
    // все методы генерируются из байткода через ASM framework
    @Override public void calcSize(T obj, CalcSizeStream css) throws IOException;
    @Override public void write(T obj, DataStream out) throws IOException;
    @Override public T read(DataStream in) throws IOException, ClassNotFoundException;
    @Override public void skip(DataStream in) throws IOException, ClassNotFoundException;
    @Override void toJson(T obj, StringBuilder builder) throws IOException;
    ...
```

Внутри используется MagicAccessorImpl или Unsafe персонально для класса.

Основная фишка библиотеки One Nio

Для каждого нестандартного класса генерируется его личный сериализатор:

```
class GeneratedSerializer … {
  // для CustomClass
  ...
```


```
class CustomClass … {
  public transient int fieldA;
  private String fieldB;
  private final List<String> fieldC;
  ...
```

Основная фишка библиотеки One Nio

Для каждого нестандартного класса генерируется его личный сериализатор:

```
class GeneratedSerializer … {
  // для CustomClass
  ...
```

знает всё

о полях

```
class CustomClass … {
  public transient int fieldA;
  private String fieldB;
  private final List<String> fieldC;
  ...
```

Основная фишка библиотеки One Nio

Для каждого нестандартного класса генерируется его личный сериализатор:

```
class GeneratedSerializer … {
  // для CustomClass
  write() быстро читает поля
  read() быстро вставляет поля
  ...
```

знает всё о полях

```
class CustomClass … {
  public transient int fieldA;
  private String fieldB;
  private final List<String> fieldC;
  ...
```

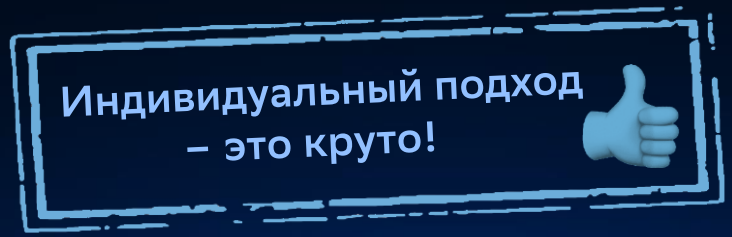
Основная фишка библиотеки One Nio

Для каждого нестандартного класса генерируется его личный сериализатор:

```
class GeneratedSerializer … {
  // для CustomClass
  write() быстро читает поля
  read() быстро вставляет поля
  ...
```

знает всё → о полях

```
class CustomClass … {
  public transient int fieldA;
  private String fieldB;
  private final List<String> fieldC;
  ...
```



План

1. *Что за One Nio и почему это круто*
2. **Проблема с кастомными методами Java-сериализации**
3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)
4. И что со скоростью? Разгон сериализации

One Nio и кастомная Java-сериализация

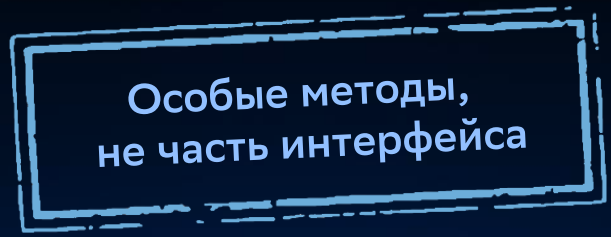
One Nio не работает с кастомными методами Java-сериализации:

```
private void writeObject(ObjectOutputStream out) throws IOException
private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundxception
private Object writeReplace() throws ObjectStreamException
private Object readResolve() throws ObjectStreamException
```

One Nio и кастомная Java-сериализация

One Nio не работает с кастомными методами Java-сериализации:

```
private void writeObject(ObjectOutputStream out) throws IOException
private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundxception
private Object writeReplace() throws ObjectStreamException
private Object readResolve() throws ObjectStreamException
```



Классы с кастомными методами сериализации

В различных библиотеках, например, в Guava

- `ImmutableList`
- `ImmutableSet`
- `ImmutableMap`

SBER TECH

Классы с кастомными методами сериализации

В различных библиотеках, например, в Guava

- `ImmutableList`
- `ImmutableSet`
- `ImmutableMap`

В JDK таких классов всё больше и больше

- Java 1+:
  `InetAddress` и `InetSocketAddress`
- Java 8+:
  Date Time API из пакета `java.time.*`
- Java 9+:
  `List.of()`, `Set.of()` и `Map.of()`
- и т.д.

Кастомная сериализация на примере List.of()

```
class ImmutableCollections … {
  class List12 … {
    private final E e0;
    private final Object e1;
    private Object writeReplace() {
      if (e1 == EMPTY)
        return new CollSer(…, e0);
      else
        return new CollSer(…, e0, e1);
    }
  ...
```

Кастомная сериализация на примере List.of()

```
class ImmutableCollections … {
  class List12 … {
    private final E e0;
    private final Object e1;
    private Object writeReplace() {
      if (e1 == EMPTY)
        return new CollSer(…, e0);
      else
        return new CollSer(…, e0, e1);
    }
  ...
```

делегируется

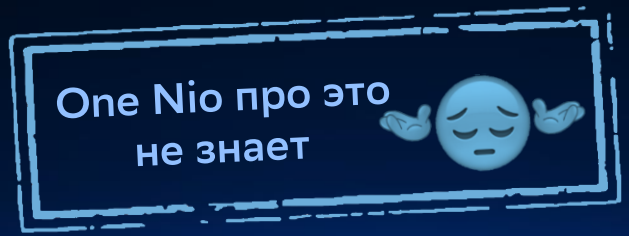
```
class ImmutableCollections … {
  class CollSer … {
    private void writeObject(…) … {
      // запись байтов в поток
    }
    private void readObject(…) … {
      // чтение байтов из потока
    }
    private Object readResolve() … {
      // CollSer -> List12
    }
  ...
```

Кастомная сериализация на примере List.of()

```
class ImmutableCollections … {
  class List12 … {
    private final E e0;
    private final Object e1;
    private Object writeReplace() {
      if (e1 == EMPTY)
        return new CollSer(…, e0);
      else
        return new CollSer(…, e0, e1);
    }
  ...
```

делегируется

```
class ImmutableCollections … {
  class CollSer … {
    private void writeObject(…) … {
      // запись байтов в поток
    }
    private void readObject(…) … {
      // чтение байтов из потока
    }
    private Object readResolve() … {
      // CollSer -> List12
    }
  ...
```



One Nio сериализация на примере List.of()

Список из 1 элемента

```
class ImmutableCollections … {
  class List12 … {
    private final E e0;
    private final Object e1;

    List12(E e0) {
      this.e0 = e0;
      this.e1 = ImmutableCollections.EMPTY;
    }
  ...
```

One Nio сериализация на примере List.of()

Список из 1 элемента

```
class ImmutableCollections … {
  class List12 … {
    private final E e0;
    private final Object e1;

    List12(E e0) {
      this.e0 = e0;
      this.e1 = ImmutableCollections.EMPTY;
    }
  ...
```

```
class OneNioUtils {
  public byte[] serialize(Object obj) {
    // запись полей e0 и e1 в поток
  }
  public Object deserialize(byte[] bytes){
    // чтение полей e0 и e1 из потока
  }
}
```

One Nio сериализация на примере List.of()

Список из 1 элемента

```
class ImmutableCollections … {
  class List12 … {
    private final E e0;
    private final Object e1;

    List12(E e0) {
      this.e0 = e0;
      this.e1 = ImmutableCollections.EMPTY;
    }
  ...
```

```
class OneNioUtils {
  public byte[] serialize(Object obj) {
    // запись полей e0 и e1 в поток
  }
  public Object deserialize(byte[] bytes){
    // чтение полей e0 и e1 из потока
  }
}
```

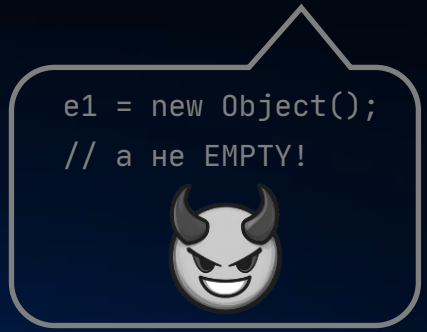
```
e1 = new Object();
// а не EMPTY!
```

One Nio сериализация на примере List.of()

Список из 1 элемента

```
List<SomeClass> list1 = List.of(element);
byte[] bytes = OneNioUtils.serialize(list);
List<SomeClass> list2 =
    OneNioUtils.deserialize(bytes);
list2.size() == 2!
list2.get(1).getClass() == Object.class
```

```
class OneNioUtils {
  public byte[] serialize(Object obj) {
    // запись полей e0 и e1 в поток
  }
  public Object deserialize(byte[] bytes){
    // чтение полей e0 и e1 из потока
  }
}
```



One Nio сериализация на примере List.of()

Список из 1 элемента

```
List<SomeClass> list1 = List.of(element);
byte[] bytes = OneNioUtils.serialize(list);
List<SomeClass> list2 =
    OneNioUtils.deserialize(bytes);
list2.size() == 2!
list2.get(1).getClass() == Object.class
```

Ошибки после десериализации

```
class OneNioUtils {
  public byte[] serialize(Object obj) {
    // запись полей e0 и e1 в поток
  }
  public Object deserialize(byte[] bytes){
    // чтение полей e0 и e1 из потока
  }
}
```

```
e1 = new Object();
// а не EMPTY!
```

Неутешительные выводы

Нельзя сериализовывать объект по его полям

Если класс использует кастомную логику Java-сериализации.

Риски undefined behaviour при сериализации One Nio

Каждый такой случай уникален.

Неутешительные выводы

Нельзя сериализовывать объект по его полям

Если класс использует кастомную логику Java-сериализации.

Риски undefined behaviour при сериализации One Nio

Каждый такой случай уникален.

Надо что-то с этим делать!

План

1. *Что за One Nio и почему это круто*
2. *Проблема с кастомными методами Java-сериализации*
3. **«Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)**
4. И что со скоростью? Разгон сериализации

План

- ✅ *1. Что за One Nio и почему это круто*
- ✅ *2. Проблема с кастомными методами Java-сериализации*
- ✅ **3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)**
  - 3.1. Как реализовали defaultWriteObject() и defaultReadObject()?
  - 3.2. Как обеспечили десериализацию зацикленных структур данных?
  - 3.3. Как обошли ограничения JPMS?
- 4. И что со скоростью?

С чего мы начали

С очевидного:

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {
    @Override public void calcSize(T obj, CalcSizeStream css) throws IOException;
    @Override public void write(T obj, DataStream out) throws IOException;
    @Override public T read(DataStream in) throws IOException, ClassNotFoundException;
    @Override public void skip(DataStream in) throws IOException, ClassNotFoundException;
    @Override void toJson(T obj, StringBuilder builder) throws IOException;
    ...
```

С чего мы начали

С очевидного:

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {
    @Override public void calcSize(T obj, CalcSizeStream css) throws IOException;
    @Override public void write(T obj, DataStream out) throws IOException;
    @Override public T read(DataStream in) throws IOException, ClassNotFoundException;
    @Override public void skip(DataStream in) throws IOException, ClassNotFoundException;
    @Override void toJson(T obj, StringBuilder builder) throws IOException;
    ...
```

Используется, когда в классе есть writeObject(), readObject(), writeReplace() или readResolve().

Что сделали в первую очередь

Реализовали ObjectOutputStream и ObjectInputStream с One Nio «под капотом».

```
private void writeObject(ObjectOutputStream out) throws IOException
private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundxception
private Object writeReplace() throws ObjectStreamException
private Object readResolve() throws ObjectStreamException
```

Что сделали в первую очередь

```
class DataObjectOutputStream … {
  ...
  @Override public void writeBoolean(boolean v)…{
    oneNioStream.writeBoolean(v);
  }
  @Override public void writeByte(int v)…{
    oneNioStream.writeByte(v);
  }
  @Override public void writeShort(int v)…{
    oneNioStream.writeShort(v);
  }
  ...
  @Override public void defaultWriteObject()…{
    // ???
  }
```

```
class DataObjectInputStream … {
  ...
  @Override public boolean readBoolean()…{
    return oneNioStream.readBoolean();
  }
  @Override public byte readByte()…{
    return oneNioStream.readByte();
  }
  @Override public short readShort()…{
    return oneNioStream.readShort();
  }
  ...
  @Override public void defaultReadObject()…{
    // ???
  }
```

План

1. *Что за One Nio и почему это круто*

2. *Проблема с кастомными методами Java-сериализации*

3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)

   3.1. Как реализовали defaultWriteObject() и defaultReadObject()?

   3.2. Как обеспечили десериализацию зацикленных структур данных?

   3.3. Как обошли ограничения JPMS?

4. И что со скоростью? Разгон сериализации

Магия default(Write|Read)Object()

```
class ImmutableCollections … {
  class CollSer … {  // возвращается из List12.writeReplace()
    private final int tag;
    private transient Object[] array;

    private void writeObject(ObjectOutputStream stream) throws IOException {
      stream.defaultWriteObject();
      stream.writeInt(array.length);
      for (int i = 0; i < array.length; i++)
        stream.writeObject(array[i]);
    }

    private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {
      stream.defaultReadObject();
      int len = stream.readInt(); Object[] a = new Object[len];
      for (int i = 0; i < len; i++)
        a[i] = stream.readObject();
      array = a;
    }
  ...
```

Магия default(Write|Read)Object()

```
class ImmutableCollections … {
  class CollSer … {  // возвращается из List12.writeReplace()
    private final int tag;
    private transient Object[] array;

    private void writeObject(ObjectOutputStream stream) throws IOException {
      stream.defaultWriteObject();
      stream.writeInt(array.length);
      for (int i = 0; i < array.length; i++)
        stream.writeObject(array[i]);
    }

    private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {
      stream.defaultReadObject();
      int len = stream.readInt(); Object[] a = new Object[len];
      for (int i = 0; i < len; i++)
        a[i] = stream.readObject();
      array = a;
    }
  ...
```

Магия default(Write|Read)Object()

```
class ImmutableCollections … {
  class CollSer … {  // возвращается из List12.writeReplace()
    private final int tag;
    private transient Object[] array;

    private void writeObject(ObjectOutputStream stream) throws IOException {
      stream.defaultWriteObject(); // в stream попадает поле tag!
      stream.writeInt(array.length);
      for (int i = 0; i < array.length; i++)
        stream.writeObject(array[i]);
    }

    private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {
      stream.defaultReadObject();
      int len = stream.readInt(); Object[] a = new Object[len];
      for (int i = 0; i < len; i++)
        a[i] = stream.readObject();
      array = a;
    }
  ...
```

Магия default(Write|Read)Object()

```
class ImmutableCollections … {
  class CollSer … {  // возвращается из List12.writeReplace()
    private final int tag;
    private transient Object[] array;

    private void writeObject(ObjectOutputStream stream) throws IOException {
      stream.defaultWriteObject(); // в stream попадает поле tag!
      stream.writeInt(array.length);
      for (int i = 0; i < array.length; i++)
        stream.writeObject(array[i]);
    }

    private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {
      stream.defaultReadObject();  // из stream-а восстанавливается поле tag!
      int len = stream.readInt(); Object[] a = new Object[len];
      for (int i = 0; i < len; i++)
        a[i] = stream.readObject();
      array = a;
    }
  ...
```

Магия default(Write|Read)Object()

```
class ImmutableCollections … {
  class CollSer … {  // возвращается из List12.writeReplace()
    private final int tag;
    private transient Object[] array;

    private void writeObject(ObjectOutputStream stream) throws IOException {
      stream.defaultWriteObject(); // в stream попадает поле tag!
      stream.writeInt(array.length);
      for (int i = 0; i < array.length; i++)
        stream.writeObject(array[i]);
    }

    private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException {
      stream.defaultReadObject();  // из stream-а восстанавливается поле tag!
      int len = stream.readInt(); Object[] a = new Object[len];
      for (int i = 0; i < len; i++)
        a[i] = stream.readObject();
      array = a;
    }
  ...
```

Повторение default-магии. Шаг первый

Нужна такая же field-by-field сериализация

Причем transient-поля она трогать не должна.

GeneratedSerializer?

Сериализует по полям, учитывает transient.

Повторение default-магии. Шаг первый

Нужна такая же field-by-field сериализация

Причем transient-поля она трогать не должна.

GeneratedSerializer?

Сериализует по полям, учитывает transient.

Подойдёт для defaultWriteObject() и defaultReadObject()

Повторение default-магии. Нюанс

GeneratedSerializer сам вызывает writeObject() и readObject()

```
List<SomeClass> list1 = List.of(element);
byte[] bytes = OneNioUtils.serialize(list);

    // class CollSer
    private void writeObject(ObjectOutputStream stream) throws IOException {
      stream.defaultWriteObject();

        // class DataObjectOutputStream
        @Override public void defaultWriteObject() throws IOException {
          generatedSerializer.write(collSer, oneNioStream);

Рекурсия!       // class CollSer
                private void writeObject(ObjectOutputStream stream) throws IOException {
                  stream.defaultWriteObject();
                  stream.writeInt(array.length);  // Exception!
```

Повторение default-магии. Напильник

```
public class GeneratedSerializer extends one.nio.serial.Serializer<Object> {
  GeneratedSerializer(Class<?> cls,
                      Boolean skipWriteObject, Boolean skipReadObject) {
    ...
  }
  @Override public void write(T obj, DataStream out) throws IOException {
    // если skipWriteObject, то writeObject() не вызывается
  }
  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    // если skipReadObject, то readObject() не вызывается
  }
```

Повторение default-магии. Есть инструмент

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {
  /**
   * For handle ObjectInputStream.defaultReadObject()
   * and ObjectOutputStream.defaultWriteObject()
   */
  private GeneratedSerializer defaultSerializer;

  public CustomizedSerializer(Class<?> cls, ...) {
     ...
     this.defaultSerializer = new GeneratedSerializer(cls, true, true);
  }
  ...
```

Повторение default-магии. Шаг второй

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {

  @Override public void write(T obj, DataStream out) throws IOException {
    ObjectOutputStream outStream = new DataObjectOutputStream(out,
            () -> defaultSerializer.write(obj, out));  // defaultWriteObject
    customMethods.writeObject.invoke(obj, outStream);
  }
  ...
```

Повторение default-магии. Шаг второй

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = /* создание пустого объекта */;
    ObjectInputStream inStream = new DataObjectInputStream(in, result.getClass(),
            () -> {  // defaultReadObject
               Object obj = defaultSerializer.read(in);
               // копирование obj в result
            }
    );
    customMethods.readObject.invoke(result, inStream);
    return result;
  }
  ...
```

Повторение default-магии. Шаг второй

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {

 @Override public void write(T obj, DataStream out) throws IOException {
    // Вызовы customMethods.writeReplace.invoke(obj)
    // как в стандартной Java-сериализации
  }

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    // Вызовы customMethods.readResolve.invoke(result)
    // как в стандартной Java-сериализации
  }
  ...
```

Повторение default-магии. Шаг второй

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {

 @Override public void write(T obj, DataStream out) throws IOException {
    // Вызовы customMethods.writeReplace.invoke(obj)
    // как в стандартной Java-сериализации
  }

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    // Вызовы customMethods.readResolve.invoke(result)
    // как в стандартной Java-сериализации
  }
  ...
```

Стандартная Java-сериализация тоже через рефлексию вызывает кастомные методы

Повторили стандартную Java-сериализацию

Задача завершена?

Добавили в One Nio CustomizedSerializer, который как надо вызывает writeObject(), readObject(), writeReplace() и readResolve().

Повторили стандартную Java-сериализацию

Задача завершена?

Добавили в One Nio CustomizedSerializer, который как надо вызывает writeObject(), readObject(), writeReplace() и readResolve().

Нет (было бы слишком просто)

Нашли целый ряд нюансов!
Не переключайтесь.

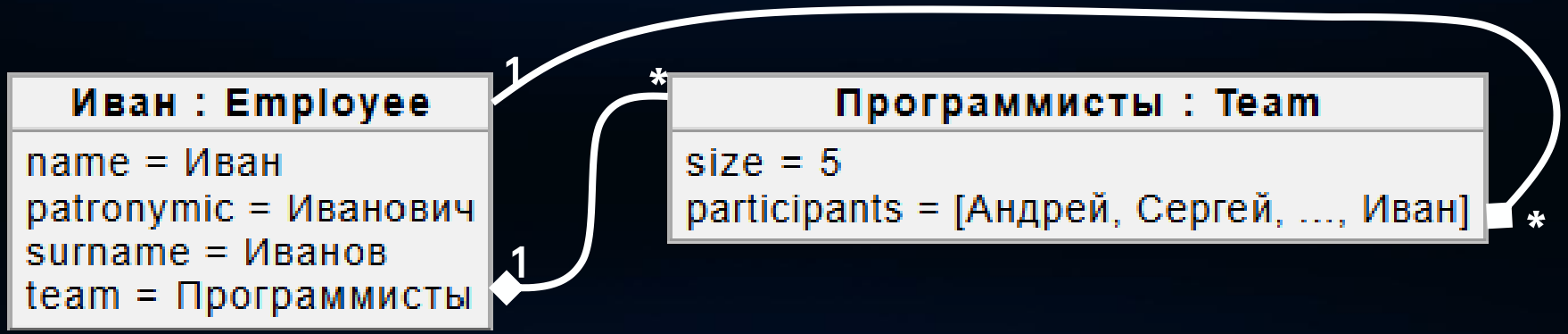
SBER TECH

План

- ✓ *1. Что за One Nio и почему это круто*
- ✓ *2. Проблема с кастомными методами Java-сериализации*
- ✓ 3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)
  - ✓ 3.1. Как реализовали defaultWriteObject() и defaultReadObject()?
  - ✓ **3.2. Как обеспечили десериализацию зацикленных структур данных?**
  - 3.3. Как обошли ограничения JPMS?
- 4. И что со скоростью? Разгон сериализации

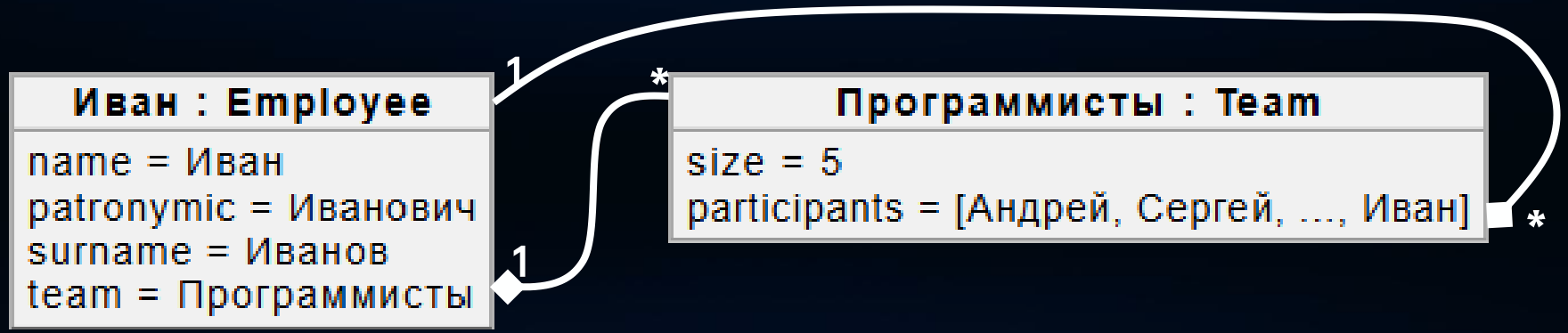
Пример зацикленной структуры данных

UML диаграмма объектов:



Пример зацикленной структуры данных

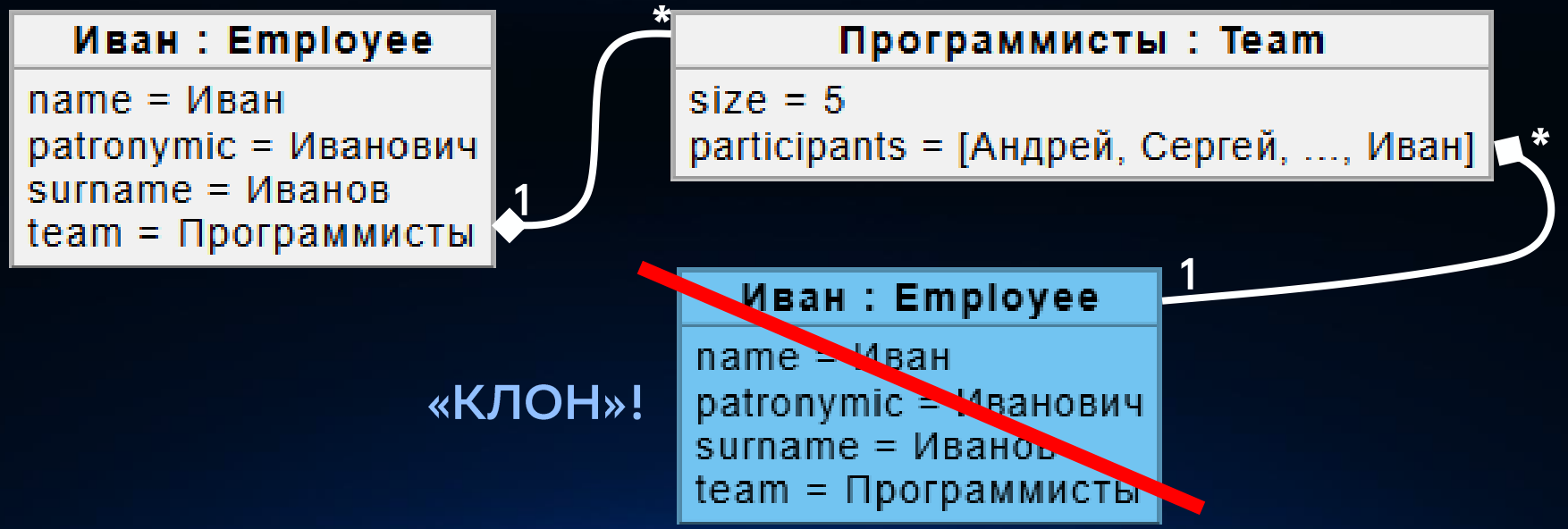
UML диаграмма объектов:



Так и должно остаться
после десериализации

Пример зацикленной структуры данных

UML диаграмма объектов:



One Nio дружит с зацикленными структурами

Но накладывается ограничение для сериализаторов:

```
public class AnySerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = /* создание пустого объекта */;
    in.register(result);  // регистрация объекта в потоке десериализации
    // чтение данных из потока "in" внутрь объекта "result"
    return result;
  }
  ...
```

One Nio дружит с зацикленными структурами

Но накладывается ограничение для сериализаторов:

```
public class AnySerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = /* создание пустого объекта */;
    in.register(result);  // регистрация объекта в потоке десериализации
    // чтение данных из потока "in" внутрь объекта "result"
    return result;
  }
  ...
```

One Nio дружит с зацикленными структурами

Но накладывается ограничение для сериализаторов:

Вместо read(), чтобы зациклить структуру

```
public class AnySerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = /* создание пустого объекта */;
    in.register(result);  // регистрация объекта в потоке десериализации
    // чтение данных из потока "in" внутрь объекта "result"!
    return result;
  }
  ...
```

One Nio дружит с зацикленными структурами

Но накладывается ограничение для сериализаторов:

Вместо read(), чтобы зациклить структуру

```
public class AnySerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = /* создание пустого объекта */;
    in.register(result);  // регистрация объекта в потоке десериализации
    // чтение данных из потока "in" внутрь объекта "result"!
    return result;
  }
  ...
```

К чему это всё?

Проблема с зацикленными структурами


```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = ...;
    in.register(result);
    if (есть writeReplace()) {
      T realResult = // readResolve() после readObject()
      // И как теперь данные из realResult засунуть внутрь result?
    } else
      // чтение данных вызовом result.readObject()
    return result;
  }
  ...
```

Проблема с зацикленными структурами

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = ...;
    in.register(result);
    if (есть writeReplace()) {
      T realResult = // readResolve() после readObject()
      // И как теперь данные из realResult засунуть внутрь result?
    } else
      // чтение данных вызовом result.readObject()
    return result;
  }
  ...
```

**Вот бы memcpy,
как в C/C++...**

Как быстро копировать Java-объекты?

```
package sun.misc;

public final class Unsafe {

  public void copyMemory(Object srcBase, long srcOffset,
                         Object destBase, long destOffset,
                         long bytes) {
    ...
  }
  ...
```

Как быстро копировать Java-объекты?

```
package sun.misc;

public final class Unsafe {

  public void copyMemory(Object srcBase, long srcOffset,
                         Object destBase, long destOffset,
                         long bytes) {
    ...
  }
  ...
```

Как пришлось копировать Java-объекты

По полям ~~по лесам~~

- Но ни какой рефлексии
- Нельзя «подставлять» One Nio

SBER TECH

Как пришлось копировать Java-объекты

По полям ~~по лесам~~

- Но ни какой рефлексии
- Нельзя «подставлять» One Nio

Пришлось заморочиться

- Попрофилировать
- Погонять benchmarks
- и мн. др.
- Получился почти «мгновенный» копировщик

Быстрое копирование полей. Часть 1

```
void copyObjectField(Object src, Object dst, Field field, Long fieldOffset) {
  Class<?> fieldCls = field.getType();
  Long fieldOffset = unsafe.objectFieldOffset(field);
  if (fieldCls == int.class)
    unsafe.putInt(dst, fieldOffset, unsafe.getInt(src, fieldOffset));
  else if (fieldCls == long.class)
    unsafe.putLong(dst, fieldOffset, unsafe.getLong(src, fieldOffset));
  ...
  else
    unsafe.putObject(dst, fieldOffset, unsafe.getObject(src, fieldOffset));
}
```

Работает даже с private и final полями!

SBER TECH

Быстрое копирование полей. Часть 2

```
Map<Class<?>, Pair<Field, Long>[]> classFieldsMap = new ConcurrentHashMap<>();
void copyObjectByFields(Class<?> srcClass, BiConsumer<Field, Long> unsafeCopyField) {
  Pair<Field, Long>[] allFields = classFieldsMap.get(srcClass);
  if (allFields != null) {
    for (Pair<Field, Long> fieldAndOffset : allFields)
      unsafeCopyField.accept(fieldAndOffset.left, fieldAndOffset.right);
  } else {
    for (Class<?> cls = srcClass; cls != Object.class; cls = cls.getSuperclass()) {
      Field[] fields = cls.getDeclaredFields();
      for (Field field : fields)
        // копирование поля через unsafeCopyField и кэширование в classFieldsMap
    }
  }
}
```

Быстрое копирование полей. Часть 2

```
Map<Class<?>, Pair<Field, Long>[]> classFieldsMap = new ConcurrentHashMap<>();
void copyObjectByFields(Class<?> srcClass, BiConsumer<Field, Long> unsafeCopyField) {
  Pair<Field, Long>[] allFields = classFieldsMap.get(srcClass); 1
  if (allFields != null) {
    for (Pair<Field, Long> fieldAndOffset : allFields)
      unsafeCopyField.accept(fieldAndOffset.left, fieldAndOffset.right);
  } else {
    for (Class<?> cls = srcClass; cls != Object.class; cls = cls.getSuperclass()) {
      Field[] fields = cls.getDeclaredFields();
      for (Field field : fields)
        // копирование поля через unsafeCopyField и кэширование в classFieldsMap
    }
  }
}
```

SBER TECH

Быстрое копирование полей. Часть 2

```
Map<Class<?>, Pair<Field, Long>[]> classFieldsMap = new ConcurrentHashMap<>();
void copyObjectByFields(Class<?> srcClass, BiConsumer<Field, Long> unsafeCopyField) {
  Pair<Field, Long>[] allFields = classFieldsMap.get(srcClass);
  if (allFields != null) {
    for (Pair<Field, Long> fieldAndOffset : allFields)
      unsafeCopyField.accept(fieldAndOffset.left, fieldAndOffset.right);
  } else {
    for (Class<?> cls = srcClass; cls != Object.class; cls = cls.getSuperclass()) {
      Field[] fields = cls.getDeclaredFields();
      for (Field field : fields)
        // копирование поля через unsafeCopyField и кэширование в classFieldsMap
    }
  }
}
```

1

2

SBER TECH

Нет проблемы с зацикленными структурами

```
public class CustomizedSerializer extends one.nio.serial.Serializer<Object> {

  @Override public T read(DataStream in) throws IOException, ClassNotFoundException {
    T result = ...;
    in.register(result);
    if (есть writeReplace()) {
      T realResult = // readResolve() после readObject()
      copyObjectByFields(realResult, result);
    } else
      // чтение данных вызовом result.readObject()
    return result;
  }
  ...
```

Решили проблему с зацикленными структурами

Задача завершена?

Справились с ограничением One Nio
на алгоритм десериализации.

SBER TECH

Решили проблему с зацикленными структурами

Задача завершена?

Справились с ограничением One Nio
на алгоритм десериализации.

Всё ещё нет

Прилетело, откуда не ждали!

SBER TECH

План

- ✓ *1. Что за One Nio и почему это круто*
- ✓ *2. Проблема с кастомными методами Java-сериализации*
- ✓ 3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)
  - ✓ 3.1. Как реализовали defaultWriteObject() и defaultReadObject()?
  - ✓ 3.2. Как обеспечили десериализацию зацикленных структур данных?
  - ✓ **3.3. Как обошли ограничения JPMS?**
- 4. И что со скоростью? Разгон сериализации

JPMS стала жестче

Жестче проверки при рефлексивном доступе

- к private-методам
- к методам классов из другого модуля (даже к public)

JPMS стала жестче

Жестче проверки при рефлексивном доступе

- к private-методам
- к методам классов из другого модуля (даже к public)

Что такое JPMS?

- Java Platform Module System появилась в Java 9.
- Модуль - это набор пакетов с исходниками (jar).
- В дескрипторе указано, каким модулям в какие пакеты есть доступ.

JPMS стала жестче

SBER TECH

Жестче проверки при рефлексивном доступе

- к private-методам
- к методам классов из другого модуля (даже к public)

Что такое JPMS?

- Java Platform Module System появилась в Java 9.
- Модуль - это набор пакетов с исходниками (jar).
- В дескрипторе указано, каким модулям в какие пакеты есть доступ.

А мы то тут причем?

- У One Nio нет доступа ко всем модулям.
- CustomizedSerializer рефлексивно вызывает writeObject(), readObject(), writeReplace() и readResolve().
- Они не входят ни в один interface – без рефлексии никак.

JPMS против рефлексии с другими модулями

Как хорошо было раньше:

```
Method method = ImmutableCollections.List12.class.getDeclaredMethod("writeReplace");
method.setAccessible(true);
method.invoke(obj);
```

JPMS против рефлексии с другими модулями

Как хорошо было раньше:

```
Method method = ImmutableCollections.List12.class.getDeclaredMethod("writeReplace");
method.setAccessible(true);
method.invoke(obj);
```

А сейчас не очень:

```
java.lang.reflect.InaccessibleObjectException: Unable to make private java.lang.Object
java.util.ImmutableCollections$List12.writeReplace() accessible: module java.base does not "opens java.util"
to unnamed module @6e4784bc
  at java.base/java.lang.reflect.AccessibleObject.throwInaccessibleObjectException(AccessibleObject.java:391)
  at java.base/java.lang.reflect.AccessibleObject.checkCanSetAccessible(AccessibleObject.java:367)
  at java.base/java.lang.reflect.AccessibleObject.checkCanSetAccessible(AccessibleObject.java:315)
  at java.base/java.lang.reflect.Method.checkCanSetAccessible(Method.java:203)
  at java.base/java.lang.reflect.Method.setAccessible(Method.java:197)
  ...
```

Заглянули «под капот»

```
package java.lang.reflect;

public final class Method extends Executable {
  @CallerSensitive
  public void setAccessible(boolean flag) {
    AccessibleObject.checkPermission();
    if (flag) checkCanSetAccessible(Reflection.getCallerClass());// Exception отсюда...
    setAccessible0(flag);                                        // ... если ты не внутри “java.base”
  }
  ...
```

Заглянули «под капот»

```
package java.lang.reflect;

public final class Method extends Executable {
  @CallerSensitive
  public void setAccessible(boolean flag) {
    AccessibleObject.checkPermission();
    if (flag) checkCanSetAccessible(Reflection.getCallerClass());// Exception отсюда...
    setAccessible0(flag);                                        // ... если ты не внутри “java.base”
  }
  ...
```

Стандартная Java-сериализация вызывает setAccessible() изнутри “java.base” – ей можно

Заглянули «под капот»

```
package java.lang.reflect;

public final class Method extends Executable {
  @CallerSensitive
  public void setAccessible(boolean flag) {
    AccessibleObject.checkPermission();
    if (flag) checkCanSetAccessible(Reflection.getCallerClass());// Exception отсюда...
    setAccessible0(flag);                                        // ... если ты не внутри “java.base”
  }
  ...

public class AccessibleObject ... {  // базовый для Method
  boolean setAccessible0(boolean flag) {
    this.override = flag;
    return flag;
  }
  boolean override;  // package-private
  ...
```

Заглянули «под капот»

```
package java.lang.reflect;

public final class Method extends Executable {
  @CallerSensitive
  public void setAccessible(boolean flag) {
    AccessibleObject.checkPermission();
    if (flag) checkCanSetAccessible(Reflection.getCallerClass());// Exception отсюда...
    setAccessible0(flag);                                        // ... если ты не внутри “java.base”
  }
  ...

public class AccessibleObject ... {  // базовый для Method
  boolean setAccessible0(boolean flag) {
    this.override = flag;
    return flag;
  }
  boolean override;  // package-private
  ...
```

Надо всего-то поменять это поле!

SBER TECH

Попробовали рефлексию против рефлексии

Попробовали «в лоб»:

```
AccessibleObject.class.getDeclaredField("override");
```

И получили неожиданный:

```
java.lang.NoSuchFieldException: override
   at java.base/java.lang.Class.getDeclaredField(Class.java:2782)
   ...
```

SBER TECH

Попробовали рефлексию против рефлексии

Попробовали «в лоб»:

```
AccessibleObject.class.getDeclaredField("override");
```

И получили неожиданный:

```
java.lang.NoSuchFieldException: override
   at java.base/java.lang.Class.getDeclaredField(Class.java:2782)
   ...
```

В Java 21+ рефлексивно недоступны поля из java.lang.reflect

Не помогает даже --add-opens Java.base/java.lang.reflect=ALL_UNNAMED

SBER TECH

Попробовали найти в интернете

Нашли способ через VarHandle:

```
VarHandle override =
        MethodHandles.privateLookupIn(AccessibleObject.class, MethodHandles.lookup())
                     .findVarHandle(AccessibleObject.class, "override", boolean.class);
override.set(method, true);
```

Попробовали найти в интернете

Нашли способ через VarHandle:

```
VarHandle override =
        MethodHandles.privateLookupIn(AccessibleObject.class, MethodHandles.lookup())
                     .findVarHandle(AccessibleObject.class, "override", boolean.class);
override.set(method, true);
```

Круто, но работает только с ключиком:

`--add-opens java.base/java.lang.reflect=ALL-UNNAMED`

Неприемлемо из-за обратной совместимости.

Не удалось перехитрить JPMS...

Попробовали через Unsafe

Unsafe умеет работать с private и final полями, в т.ч. из другого модуля:

```
Long overrideOffset = unsafe.objectFieldOffset(overrideField);
unsafe.putBoolean(accessibleObject, overrideOffset, true);
```

Попробовали через Unsafe

Unsafe умеет работать с private и final полями, в т.ч. из другого модуля:

```
Long overrideOffset = unsafe.objectFieldOffset(overrideField);
unsafe.putBoolean(accessibleObject, overrideOffset, true);
```

Но для получения offset-а нужен Field:

```
public final class Unsafe {
  public long objectFieldOffset(Field f) {
    ...
  }
  ...
```

Пошли по второму кругу...

Эврика! Смещения одинаковые

Что если смещение поля override не изменилось с прошлых версий Java?

```
Long overrideOffset = unsafe.objectFieldOffset(overrideField);
unsafe.putBoolean(accessibleObject, overrideOffset, true);
```

Эврика! Смещения одинаковые

Что если смещение поля override не изменилось с прошлых версий Java?

```
Long overrideOffset = unsafe.objectFieldOffset(overrideField);
unsafe.putBoolean(accessibleObject, overrideOffset, true);
```

Так и есть!

```
public class AccessibleObject ... {  // базовый для Method
  boolean override;                  // offset = 12
  volatile Object accessCheckCache;  // offset = 16
```

Эврика! Смещения одинаковые

Что если смещение поля override не изменилось с прошлых версий Java?

```
Long overrideOffset = unsafe.objectFieldOffset(overrideField);
unsafe.putBoolean(accessibleObject, overrideOffset, true);
```

Так и есть!

```
public class AccessibleObject ... {  // базовый для Method
  boolean override;                  // offset = 12
  volatile Object accessCheckCache;  // offset = 16
```

Так во всех версиях Java с 8 до 25 (HotSpot, Liberica, GraalVM, OpenJ9, etc.)!

Смещения одинаковые, но в Java 25 нюанс

Теперь можно включить компактное представление заголовков объектов:

```
-XX:+UseCompactObjectHeaders
```

Если включено, то смещения меньше на 4 байта:

```
public class AccessibleObject ... {  // базовый для Method
  boolean override;                  // offset = 8 (-4), но не на OpenJ9
  volatile Object accessCheckCache;  // offset = 12 (-4)
```

Вычисляем смещение поля при старте

```
static final long overrideOffset = getOverrideOffset();

static long getOverrideOffset() {
  return useCompactObjectHeaders() && !isOpenJ9() ? 8 : 12;
}

static boolean useCompactObjectHeaders() {
  RuntimeMXBean runtimeMxBean = ManagementFactory.getRuntimeMXBean();
  Set<String> jvmArgs = Set.copyOf(runtimeMxBean.getInputArguments());
  return jvmArgs.contains("-XX:+UseCompactObjectHeaders");
}

static boolean isOpenJ9() {
  return ManagementFactory.getRuntimeMXBean().getVmVendor().contains( "OpenJ9" );
}
```

Возвращаемся к JPMS и setAccessible()

```
void setAccessibleFor(Method method, boolean flag) {
  if (!method.isAccessible()) {
    unsafe.putBoolean(method, overrideOffset, flag);
    if (!method.isAccessible())
      throw new Error(
              "Cannot make the method " + method.getName() + " accessible");
  }
}
```

Работает с методами из любого модуля

Научили One Nio кастомным методам

Теперь CustomizedSerializer может так даже на Java 21 и выше:

```
Method method = ImmutableCollections.List12.class.getDeclaredMethod("writeReplace");
setAccessibleFor(method, true);
Object replaceObj = method.invoke(obj, (Object[]) null);
// аналогично для writeObject(), readObject() и readResolve()
```

Научили One Nio кастомным методам

Ну, теперь-то задача завершена?

Справились с ограничением JPMS
на рефлексивный вызов методов
из другого модуля на Java 21+.

Что ещё нужно?

Научили One Nio кастомным методам

Ну, теперь-то задача завершена?

Справились с ограничением JPMS на рефлексивный вызов методов из другого модуля.

Да! Ну, почти...

Но на сколько эффективен новый CustomizedSerializer?

План

- ✓ *1. Что за One Nio и почему это круто*
- ✓ *2. Проблема с кастомными методами Java-сериализации*
- ✓ *3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)*
  - *3.1. Как реализовали defaultWriteObject() и defaultReadObject()?*
  - *3.2. Как обеспечили десериализацию зацикленных структур данных?*
  - *3.3. Как обошли ограничения JPMS?*
- ✓ **4. И что со скоростью?**
  - 4.1. Разгон сериализации: один шаг вместо двух

CustomizedSerializer VS GeneratedSerializer

SBER TECH

Новый CustomizedSerializer

One Nio его использует,когда
в классе есть writeObject(), readObject(), writeReplace()
или readResolve().

Старый добрый GeneratedSerializer

Используется, когда в классе нет
кастомных методов сериализации.
Работает только с полями объекта.

Что будем сериализовывать?

SBER TECH

Уже знакомый список, создаваемый через List.of()

Что будем сериализовывать?

Уже знакомый список, создаваемый через List.of()

01

- Есть все 4 кастомных метода сериализации.
- Удобно для CustomizedSerializer-a.

Что будем сериализовывать?

SBER TECH

Уже знакомый список, создаваемый через List.of()

01

- Есть все 4 кастомных метода сериализации.
- Удобно для CustomizedSerializer-а.

02

- Есть 2 не статических поля.
- Удобно для GeneratedSerializer-а.

Что будем сериализовывать?

SBER TECH

Уже знакомый список, создаваемый через List.of()

01

- Есть все 4 кастомных метода сериализации.
- Удобно для CustomizedSerializer-a.

02

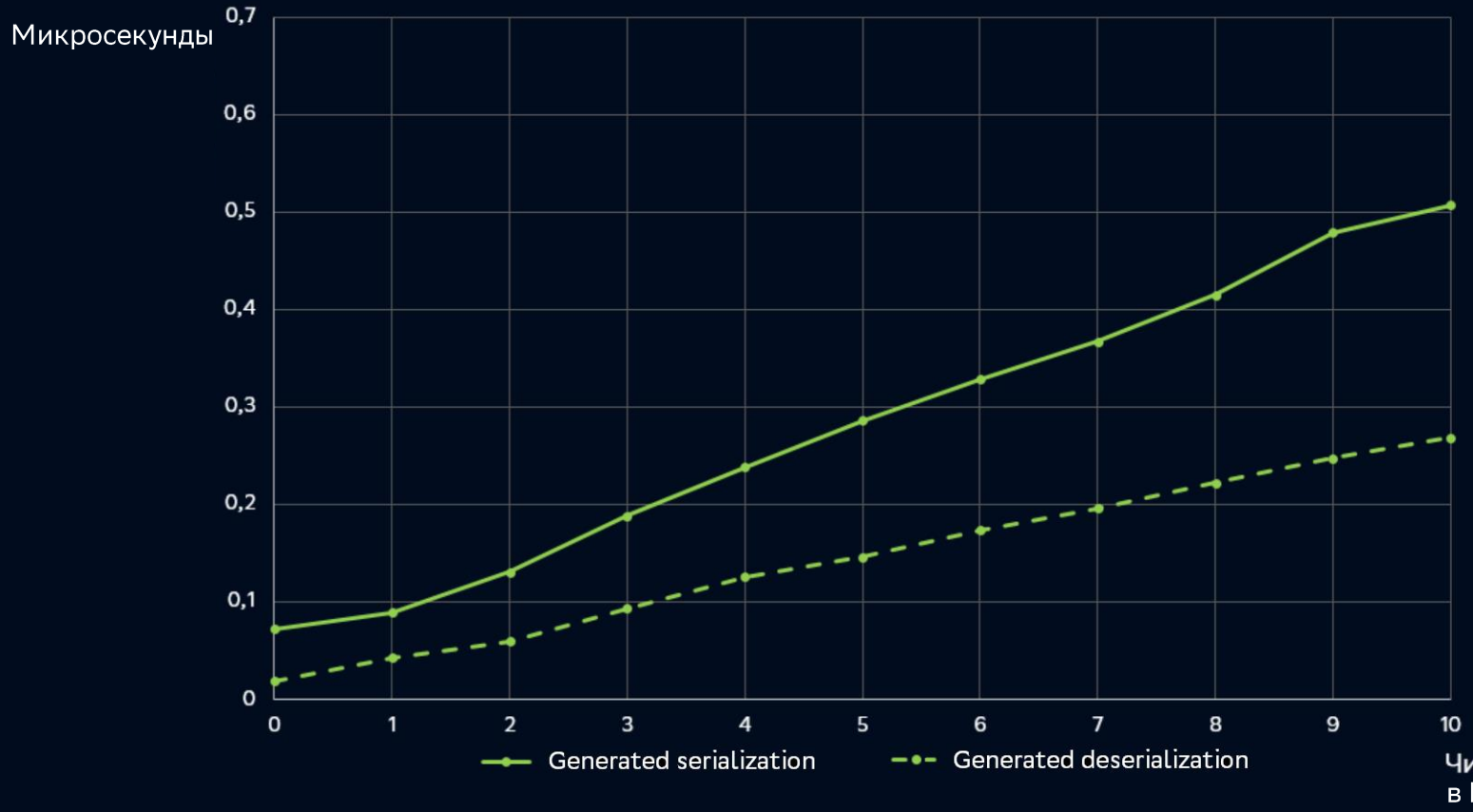
- Есть 2 не статических поля.
- Удобно для GeneratedSerializer-a.

03

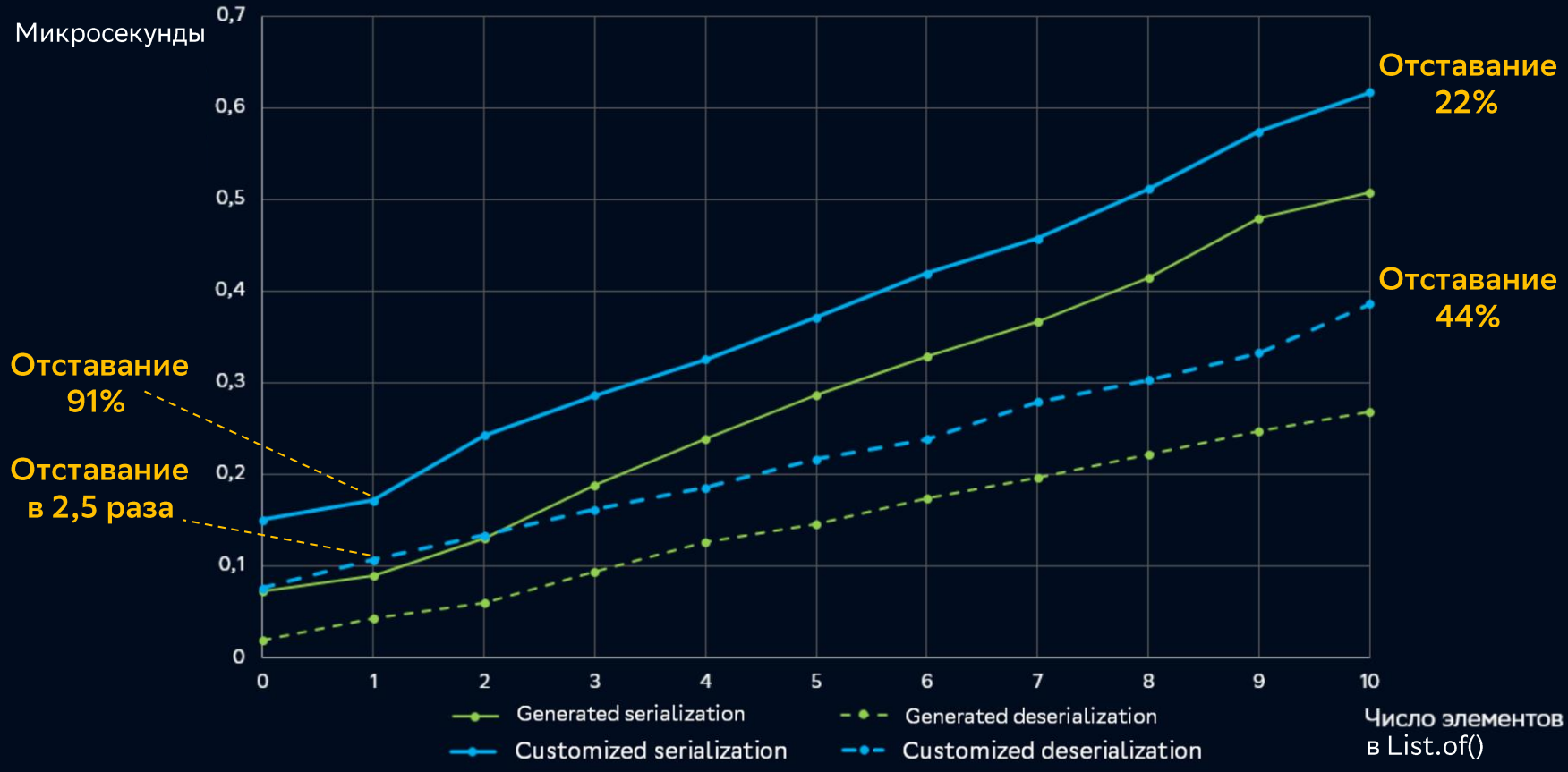
- Легко управлять размером объекта.
- Можно узнать, как это влияет на скорость.

SBER TECH

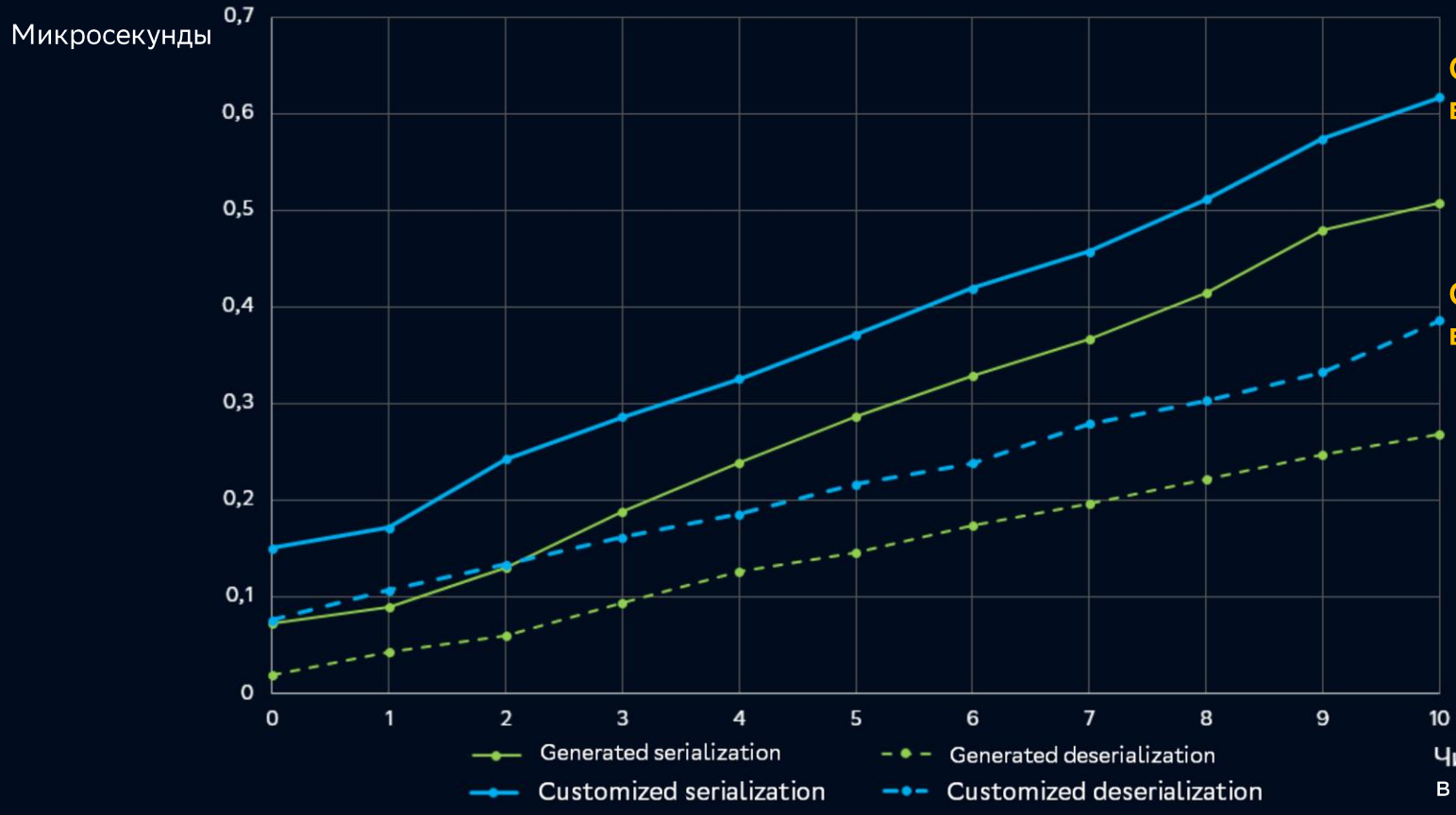
Скорость GeneratedSerializer



CustomizedSerializer VS GeneratedSerializer



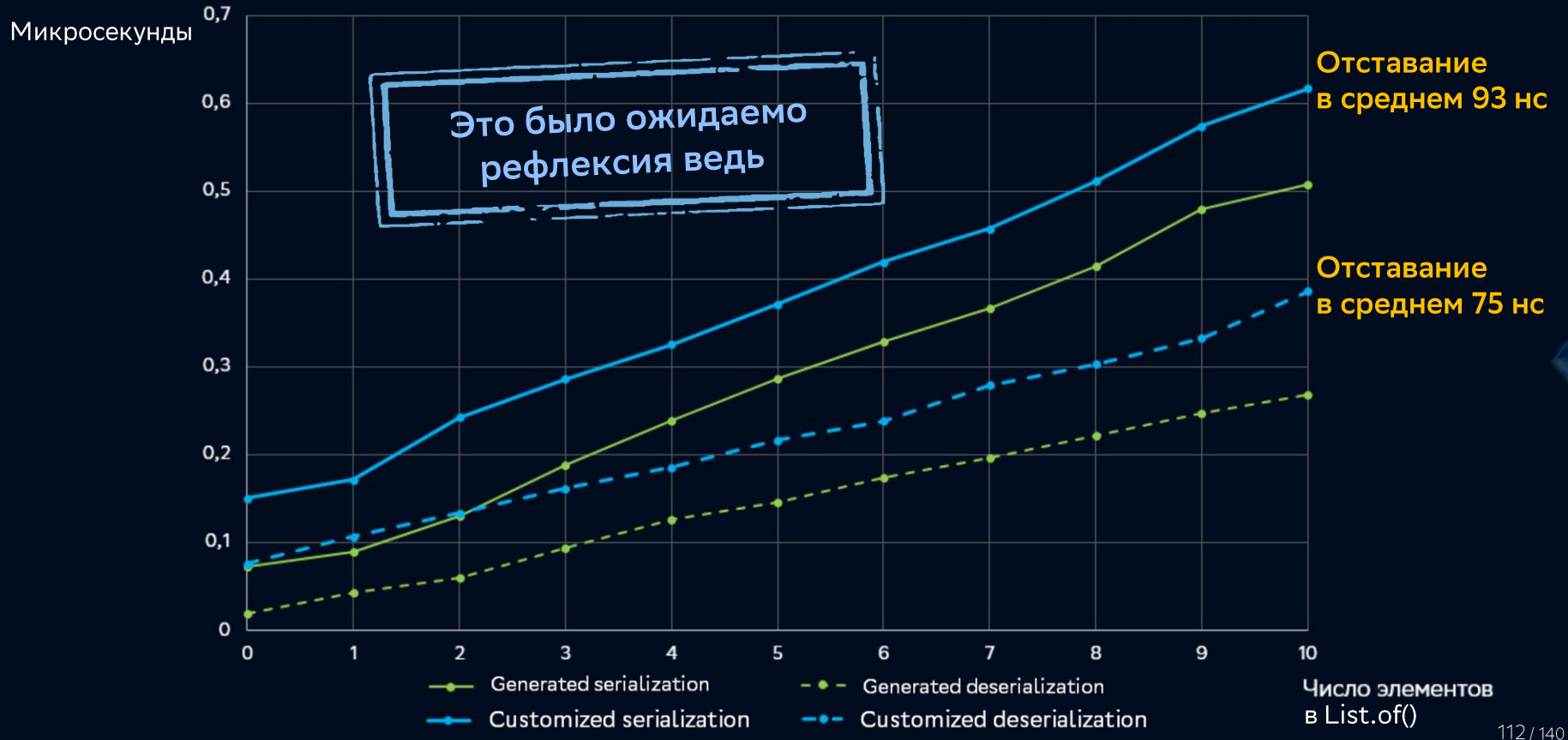
CustomizedSerializer VS GeneratedSerializer



Отставание
в среднем 93 нс

Отставание
в среднем 75 нс

CustomizedSerializer VS GeneratedSerializer

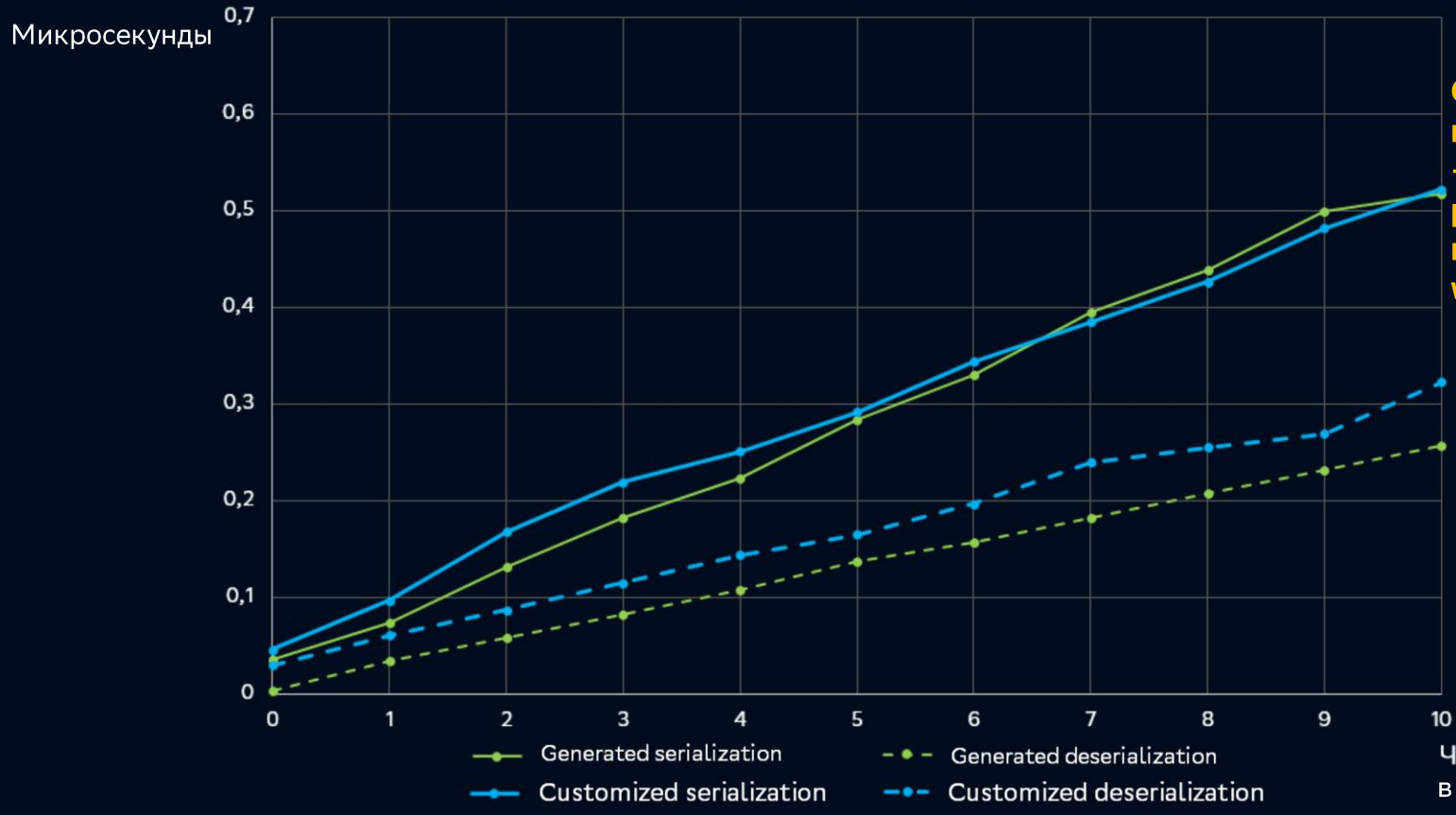


Попробуем сериализовывать POJO

```
public class PoJoCustomized implements Serializable {
  private String field1;
  private String field2;
  ...
  private String fieldN;

  // getters and setters

  private void writeObject( ObjectOutputStream out ) throws IOException {
    out.defaultWriteObject();  // внутри работает GeneratedSerializer
  }
  private void readObject( ObjectInputStream in ) throws IOException, ClassNotFoundException {
    in.defaultReadObject(); // внутри работает GeneratedSerializer
  }
}
```

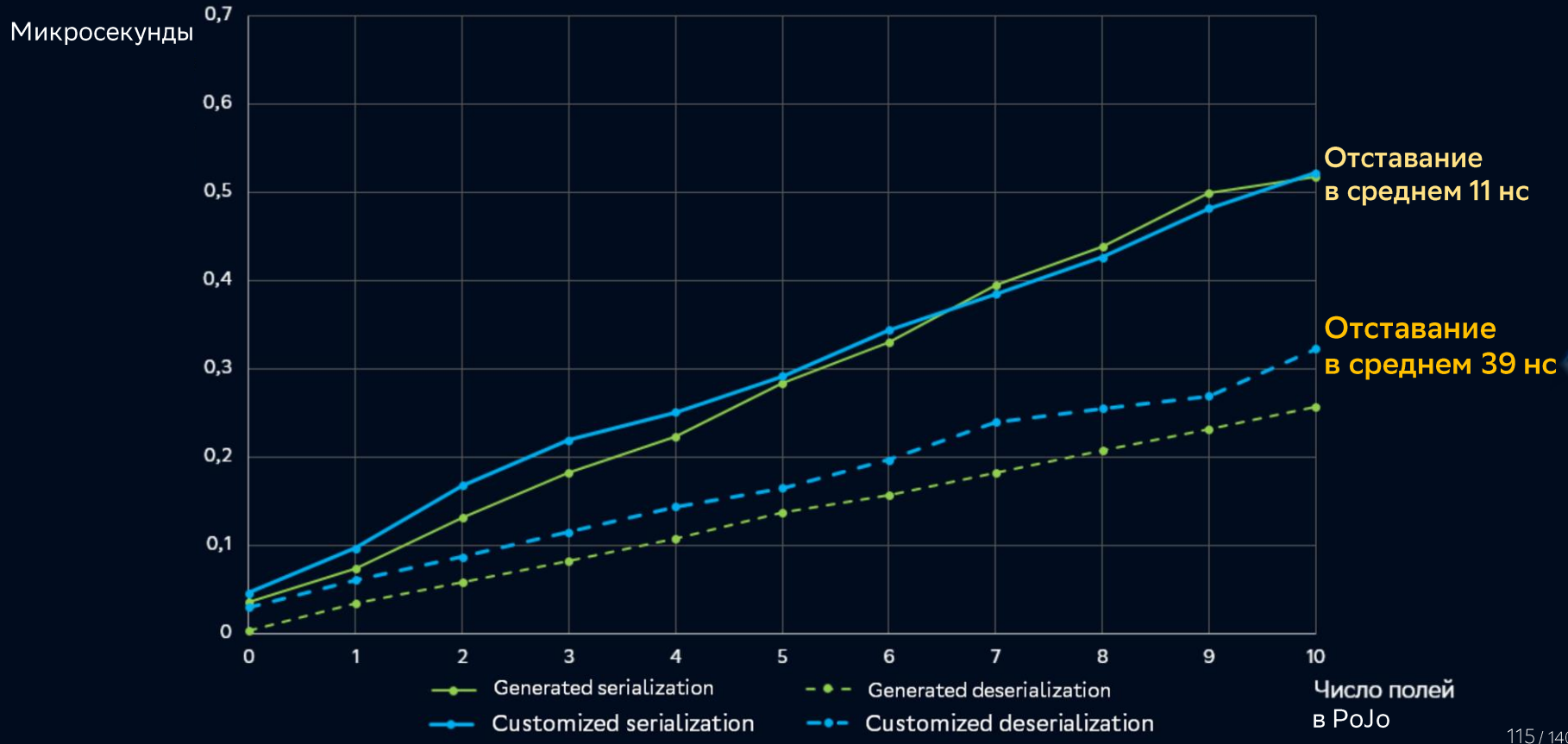
SBER TECH

Попробовали сериализовывать POJO



Отставание в среднем 11 нс - цена рефлексивного вызова writeObject()

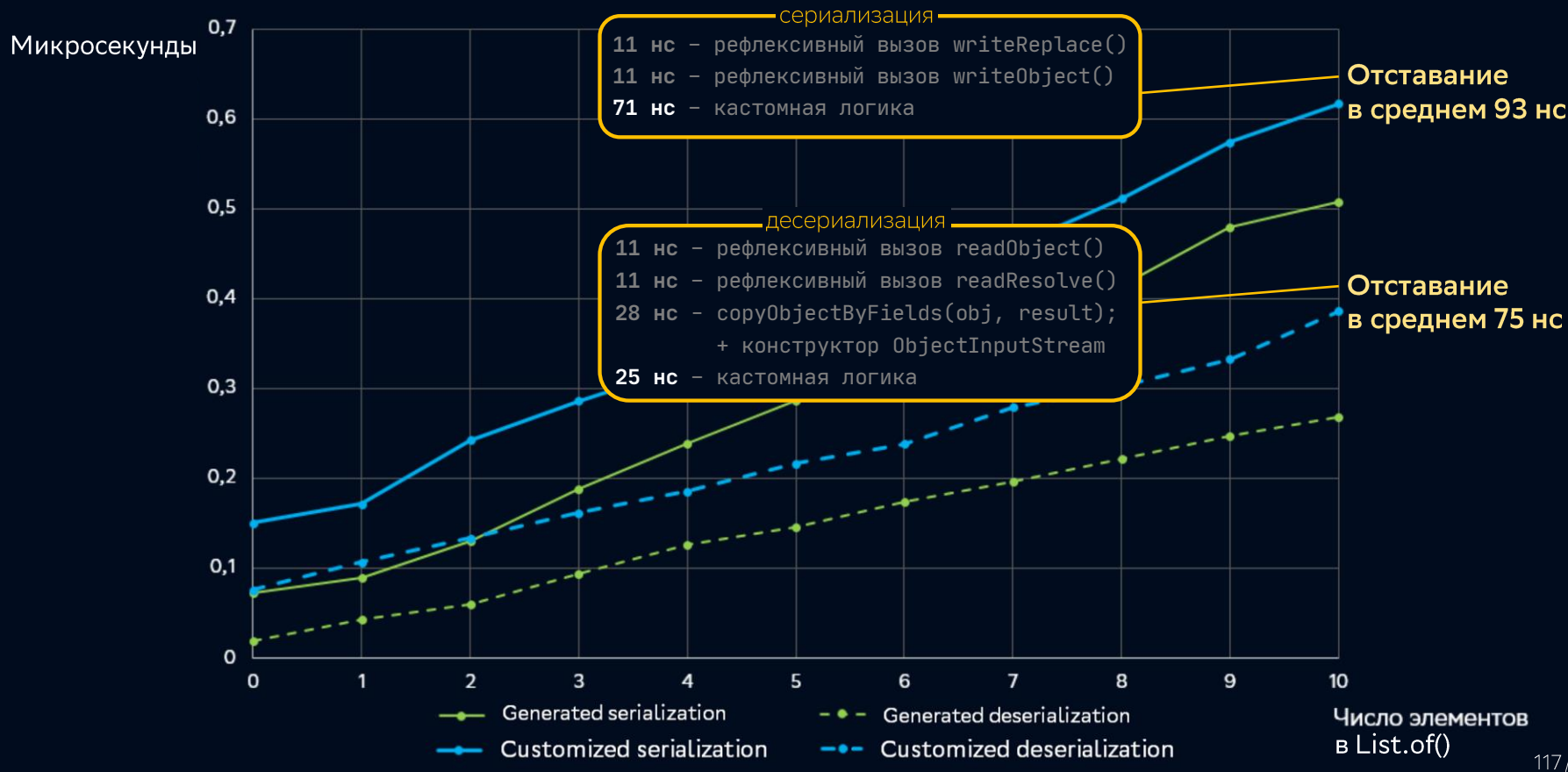
Попробовали сериализовывать POJO



SBER TECH

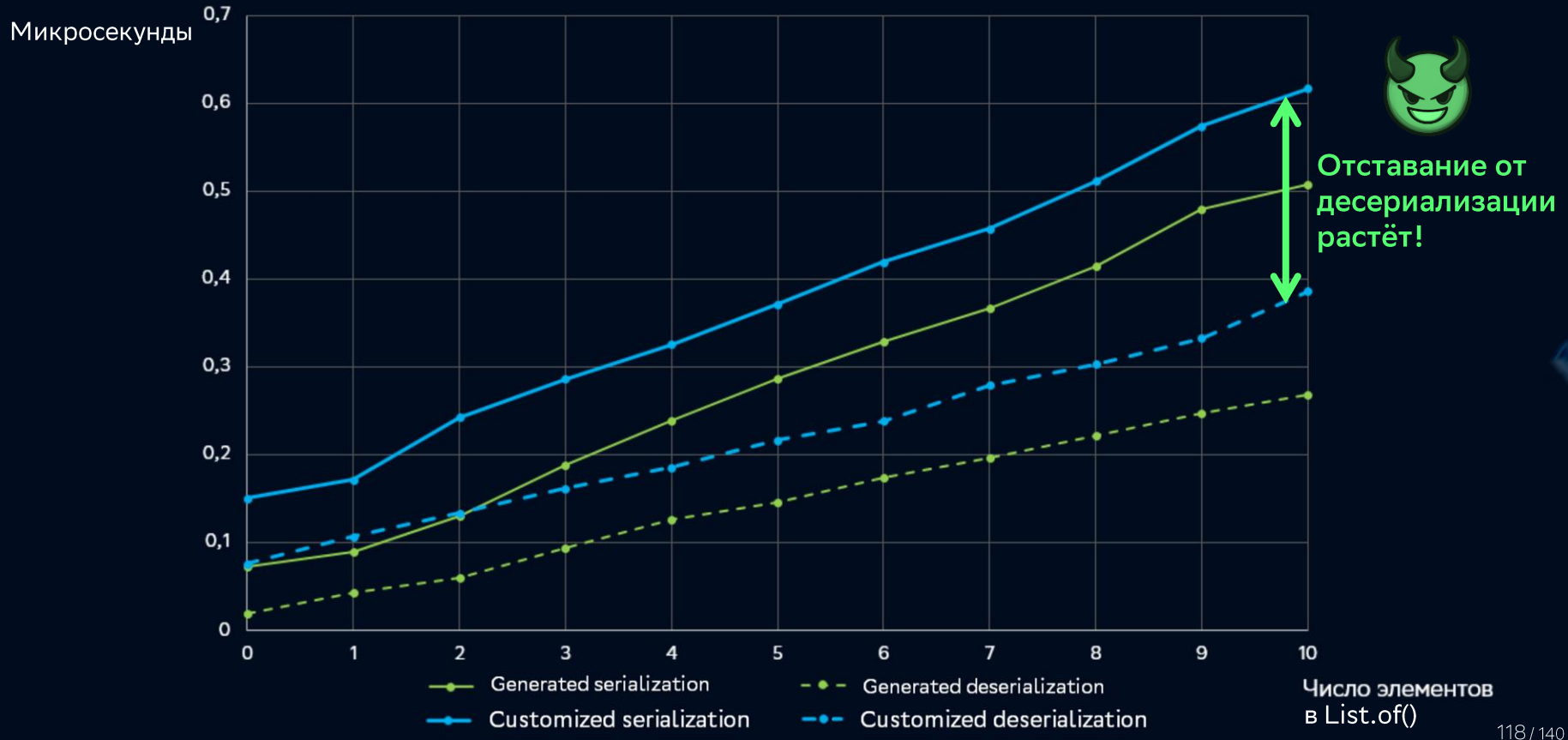
Попробовали сериализовывать POJO

CustomizedSerializer VS GeneratedSerializer



SBER TECH

Что не так с сериализацией?

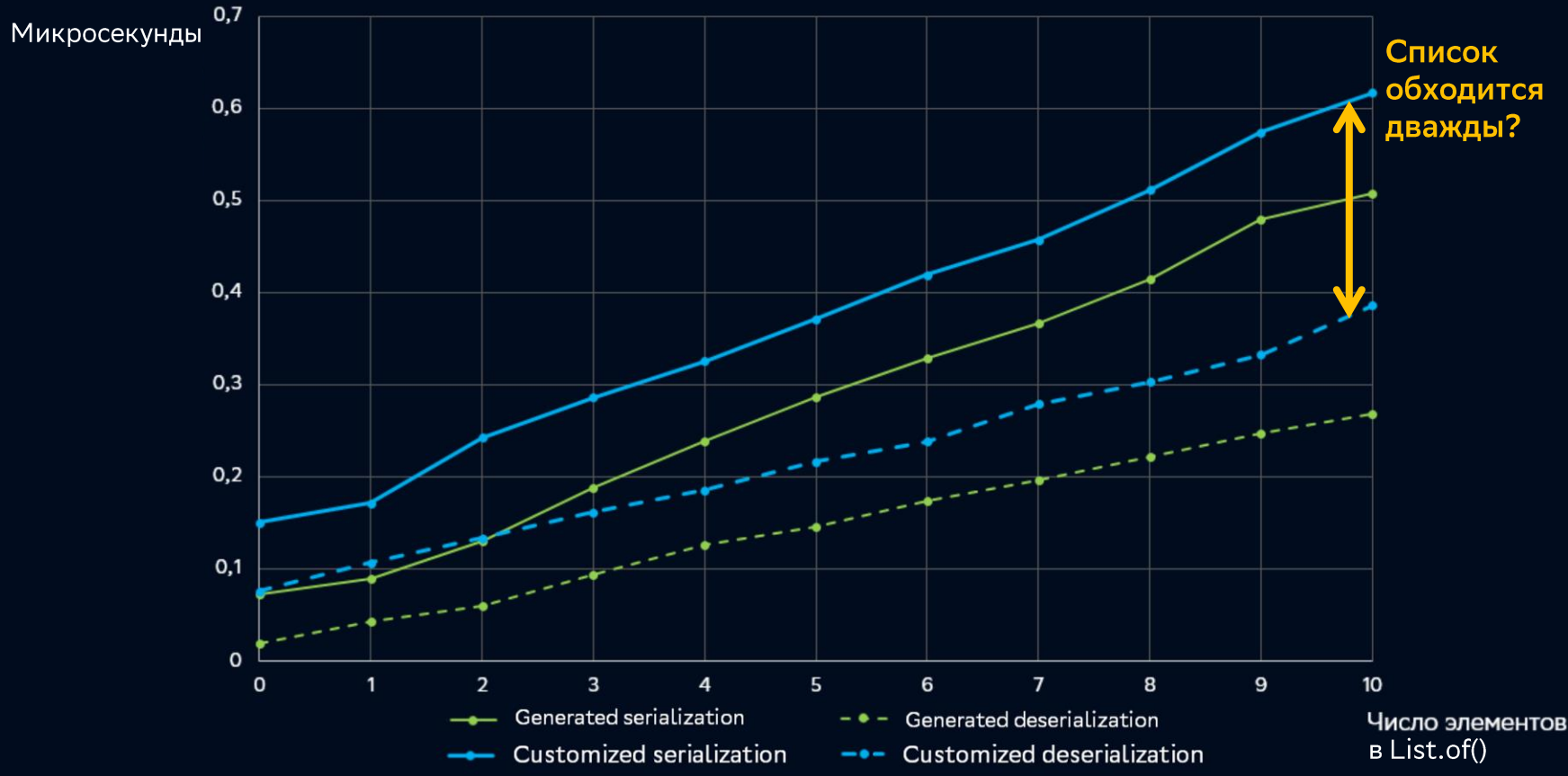


План

- ✓ *1. Что за One Nio и почему это круто*
- ✓ *2. Проблема с кастомными методами Java-сериализации*
- ✓ *3. «Повторение» в One Nio логики стандартной Java-сериализации (Java 21+)*
  - *3.1. Как реализовали defaultWriteObject() и defaultReadObject()?*
  - *3.2. Как обеспечили десериализацию зацикленных структур данных?*
  - *3.3. Как обошли ограничения JPMS?*
- ✓ 4. И что со скоростью?
- ✓ **4.1. Разгон сериализации: один шаг вместо двух**

SBER TECH

Почему время сериализации растет быстрее?



Сериализация библиотекой One Nio (as is)

```
private static <T> byte[] serialize( T object ) throws IOException {
  byte[] bufForObject = new byte[ calcBufferSizeFor( object ) ];
  try ( SerializeStream out = new SerializeStream( bufForObject ) ) {
    out.writeObject( object );
  }
  return bufForObject;
}

private static <T> int calcBufferSizeFor( T object ) throws IOException {
  try ( CalcSizeStream css = new CalcSizeStream() ) {
    css.writeObject( object );
    return css.count();
  }
}
```

Сериализация библиотекой One Nio (as is)

```
private static <T> byte[] serialize( T object ) throws IOException {
  byte[] bufForObject = new byte[ calcBufferSizeFor( object ) ];
  try ( SerializeStream out = new SerializeStream( bufForObject ) ) {
    out.writeObject( object );
  }
  return bufForObject;
}

private static <T> int calcBufferSizeFor( T object ) throws IOException {
  try ( CalcSizeStream css = new CalcSizeStream() ) {
    css.writeObject( object );
    return css.count();
  }
}
```

CalcSizeStream – меньшее из зол?

```
private static <T> byte[] serialize( T object ) throws IOException {
  byte[] bufForObject = new byte[ calcBufferSizeFor( object ) ];
  try ( SerializeStream out = new SerializeStream( bufForObject ) ) {
    out.writeObject( object );
  }
  return bufForObject;
}

private static <T> int calcBufferSizeFor( T object ) throws IOException {
  try ( CalcSizeStream css = new CalcSizeStream() ) {
    css.writeObject( object );
    return css.count();
  }
}
```

2

1

CalcSizeStream overhead

Array realloc and copy overhead

Как бы работал ByteArrayOutputStream

SBER TECH

Использовали array chain при сериализации

byte[] 1

SBER TECH

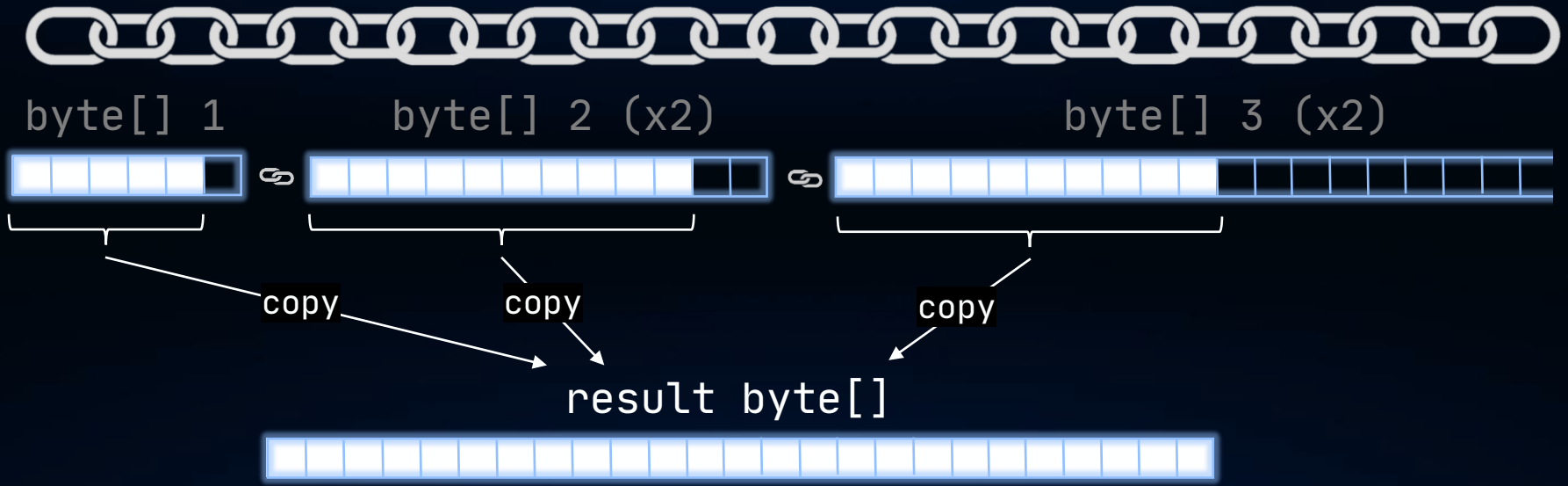
Использовали array chain при сериализации

SBER TECH

Использовали array chain при сериализации

SBER TECH

Использовали array chain при сериализации

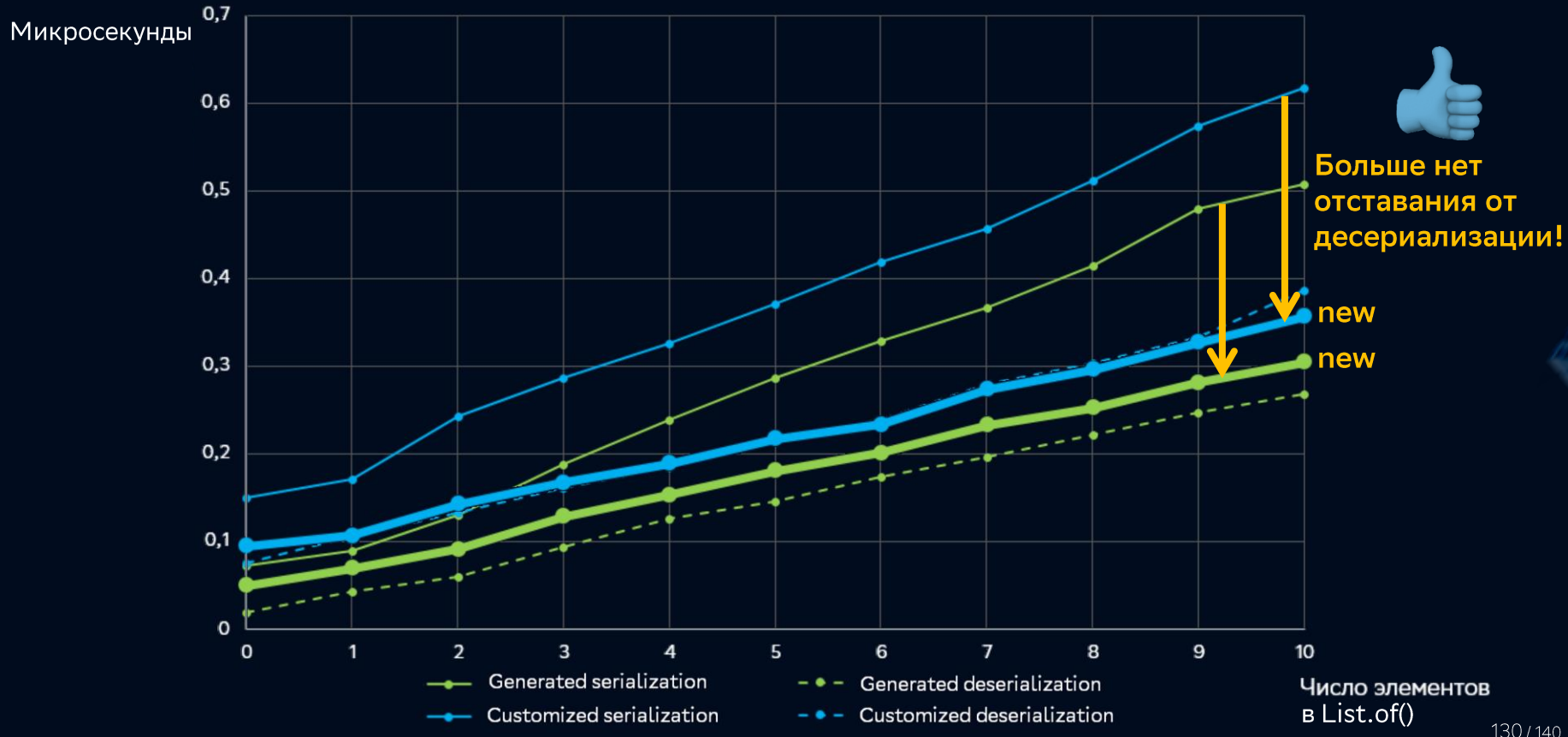


Сериализация библиотекой One Nio

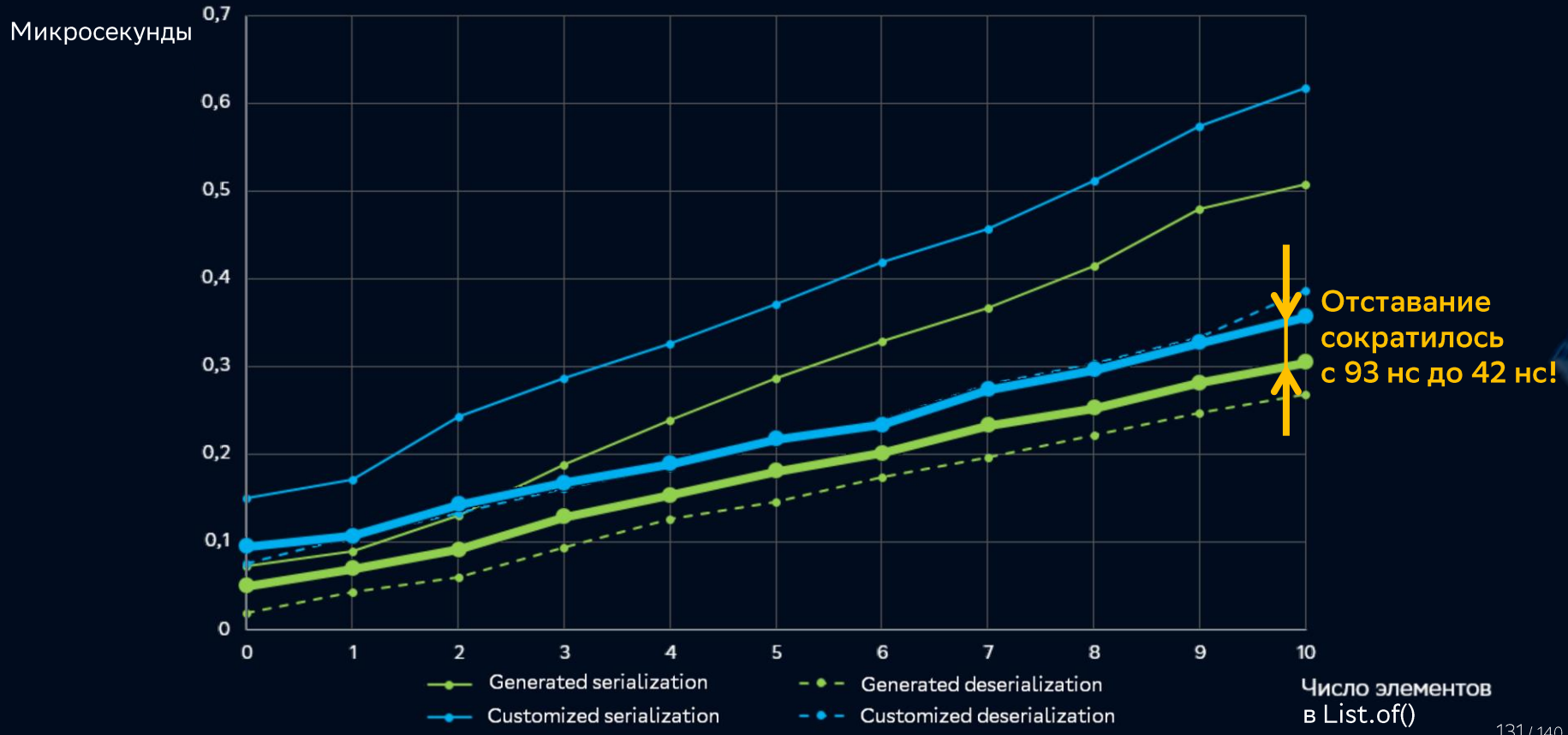
```
private static <T> byte[] serialize( T object ) throws IOException {
  byte[] bufForObject = new byte[500];
  try ( SerializeStream out = new SerializeStream( bufForObject ) ) {
    out.writeObject( object );
    return out.toByteArray();
  }
}

private static <T> int calcBufferSizeFor( T object ) throws IOException {
  try ( CalcSizeStream css = new CalcSizeStream() ) {
    css.writeObject( object );
    return css.count();
  }
}
```

SBER TECH

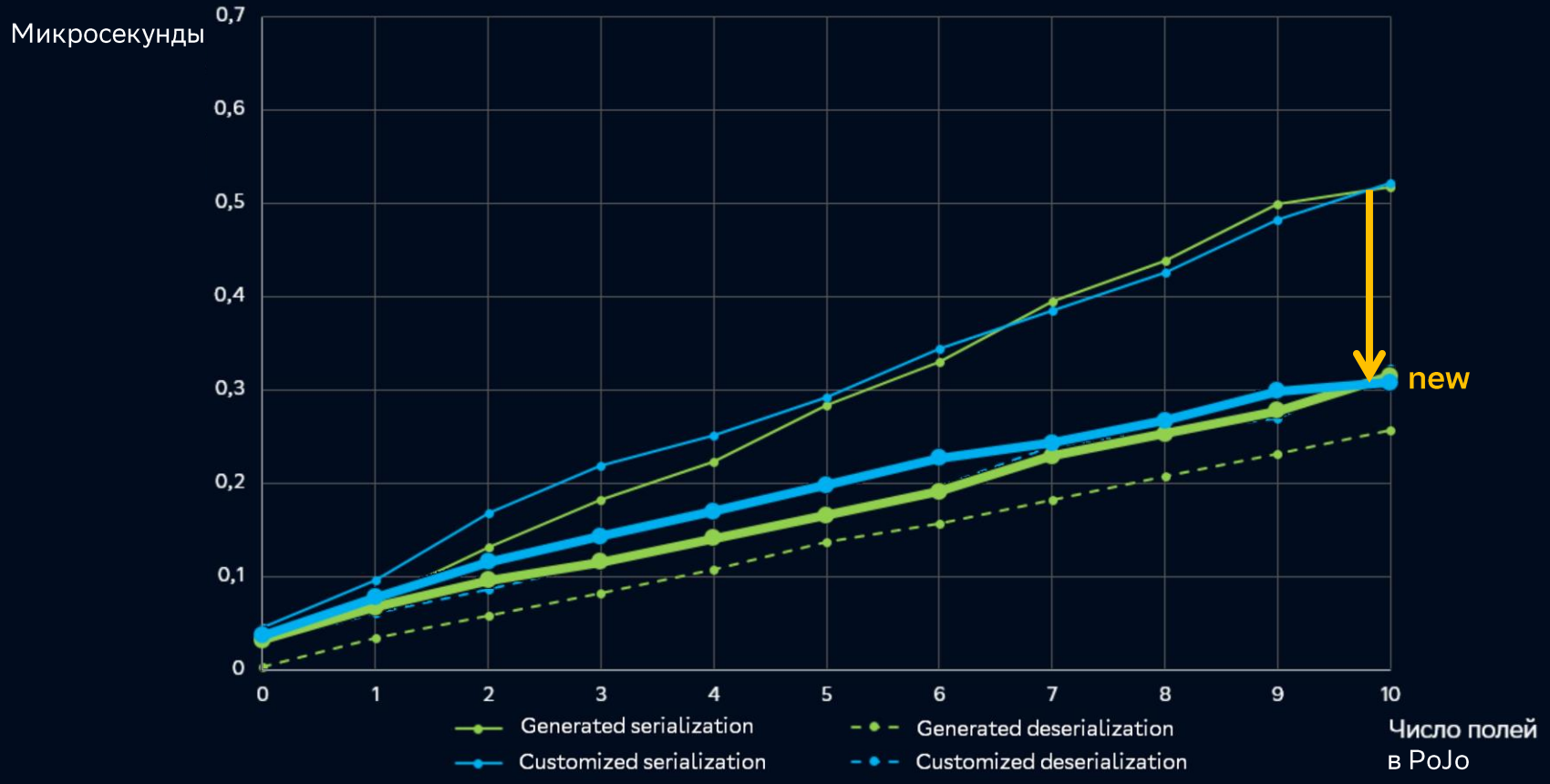
Время сериализации растет НЕ быстрее



CustomizedSerializer VS GeneratedSerializer



Сериализация POJO тоже ускорилась



Benchmark на GitHub

https://github.com/chernov-af/one-nio-benchmark

Сериализация One Nio оптимизирована

Вот теперь задача завершена!

One Nio стала быстрее сериализовывать,
не зависимо от конкретного класса.

Сами добавили overhead, сами и устранили

One Nio умеет вызывать кастомные методы:
writeObject(), readObject(), writeReplace()
и readResolve().

Выводы

Выводы

SBER TECH

01

Нужна Java-сериализация – берите One Nio, она быстрее и гибче.

Выводы

SBER TECH

01

Нужна Java-сериализация – берите One Nio, она быстрее и гибче.

02

Хотите кастомизировать сериализацию класса, но не уверены в поведении One Nio – можно больше не бояться.

Выводы

SBER TECH

01

Нужна Java-сериализация – берите One Nio, она быстрее и гибче.

02

Хотите кастомизировать сериализацию класса, но не уверены в поведении One Nio – можно больше не бояться.

03

Теперь One Nio умеет вызывать кастомные методы сериализации – можно больше не бояться новых классов с ними.

Выводы

SBER TECH

01

Нужна Java-сериализация – берите One Nio, она быстрее и гибче.

02

Хотите кастомизировать сериализацию класса, но не уверены в поведении One Nio – можно больше не бояться.

03

Теперь One Nio умеет вызывать кастомные методы сериализации – можно больше не бояться новых классов с ними.

04

Теперь One Nio сериализует ещё быстрее!

SBER TECH

Спасибо за внимание!

Андрей Чернов, Java архитектор в СберТехе
@ chernovaf@mail.ru
chernovaf